National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

University of Alberta

NETWORK PROTOCOL OVERHEAD IN DISTRIBUTED CONCURRENCY CONTROL PERFORMANCE

by

Youping Niu

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
SPRING, 1991

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-66696-X

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Youping Niu

TITLE OF THESIS: Network Protocol Overhead in Distributed Concurrency Control Performance

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: SPRING 1991

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) Youping Niu

Permanent Address:
207-10630-114 Street
Edmonton, Alberta
Canada T5H 3J9

Date: Dec 20 1990

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Network Protocol Overhead in Distributed Concurrency Control Performance** submitted by **YOUPING NIU** in partial fulfillment of the requirements for the degree of **Master of Science.**

M. T. Ozsu

(Supervisor)

P. Gburzynski

L. Y. Yuan

N. G. Narasimha Prasad

Date: Dec. 20 1990

ABSTRACT

This thesis is concerned with the effect of local area network protocols on the performance of distributed concurrency control algorithms. The issue is studied by simulating two lock-based distributed concurrency control algorithms, namely the centralized locking algorithm (CL) and the distributed locking algorithm (DL), over two local area network architectures (Ethernet and token ring). The results indicate that under heavy loads token ring has advantages over Ethernet for DL in which the network delay is very critical. Transaction performance of CL over Ethernet is highly variable and will be perceived as much less consistent than one on a ring.

Acknowledgements

I wish to thank my supervisor, Dr. M. Tamer Ozsu, for his guidance, support and advice throughout this research. I would like to thank Dr. P. Gburzynski for his invaluable assistance in getting the implementation portion of this thesis going. I would also like to thank Dr. L. Y. Yuan and Dr. N. G. Narasimha Prasad, for their helpful comments.

Thanks are also due to Chengqi Liu and Hui Lin for their support and encouragement throughout the writing of the thesis.

I am very grateful to the members of my family, especially my mother and my wife, for their unselfish and devoted support.

Finally, I would like to acknowledge the use of the LANSF simulator package developed by Dr. P. Gburzynski and Dr. P Rudnicki.

Table of Contents

Chapter 1: Introduction 1

1.1. Objective of Thesis 3

1.2. Organization of Thesis 4

Chapter 2: Concurrency Control in Distributed Database Systems 5

2.1. Terminology 5

2.2. Distributed Database Consistency Problems 7

2.3. Internal Consistency Mechanisms 9

2.4. Mutual Consistency Mechanisms 10

2.4.1. Centralized Locking 11

2.4.2. Distributed Locking 12

2.5. Deadlock 12

Chapter 3: Distributed Databases in a LAN Environment 15

3.1. LANs and their Distinct Features 15

3.2. Influence of LAN features on DDB Design 16

3.2.1. Data Fragmentation and Allocation 17

3.2.2. Distributed Query Processing 18

3.2.3. DDB Administration Issues 20

3.3. Ethernet and Token Ring Local Area Networks 21

3.3.1. Why Ethernet and Token Ring? 21

3.3.2. Ethernet and the CSMA/CD Protocol 22

3.3.3. Token Ring Architecture and Protocol 24

Chapter 4: The Simulation Model 27
4.1. Overview of The Simulation Model 27
4.2. Modeling of The Concurrency Control Algorithms 29
4.2.1. Centralized Locking Algorithm 30
4.2.1.1. Central Site Model 32
4.2.1.2. Other Site Model 36
4.2.2. Distributed Locking Algorithm 39
4.2.2.1. Model for The Distributed Locking Algorithm 41
4.3. The Local Area Network Model 43
4.3.1. The Ethernet Protocol 44
4.3.2. The Token Ring 47
4.4. Input Parameters 50
4.5. Performance Metrics 51
4.6. Design Considerations 52
4.7. Simulating Environment 53
Chapter 5: Simulation Results 54
5.1. CL over Ethernet and Token Ring 54
5.1.1. Effect of Interarrival Time 54
5.1.2. Effect of Date Rate 58
5.1.3. Effect of Number of Sites 62
5.1.4. Effect of Base Set Size 63
5.1.5. Effect of Database Size 65

5.2. DL over Ethernet and Token Ring 67

5.2.1. Effect of Interarrival Time 67

5.2.2. Effect of Date Rate 70

5.2.3. Effect of Number Sites 72

Chapter 6: Implications for DDBS Design 75

6.1. Two-Phase Commit Protocols 75

6.2. Concurrency Control Algorithms 78

6.3. Deadlock Detection/Prevention 80

6.4. Network Utilization 80

Chapter 7: Conclusions 82

Bibliography 84

Appendix A1: Performance Results 88

List of Tables

A1.1 Effect of interarrival time on mean response time. 89

A1.2 Effect of interarrival time on I/O utilization. 89

A1.3 Effect of interarrival time on CPU utilization. 89

A1.4 Effect of interarrival time on mean throughput. 90

A1.5 Effect of interarrival time on mean conflict. 90

A1.6 Effect of interarrival time on network utilization. 90

A1.7 Effect of mean Base set on mean response time. 91

A1.8 Effect of mean Base set on I/O utilization. 91

A1.9 Effect of mean Base set on CPU utilization. 91

A1.10 Effect of mean Base set on mean throughput. 92

A1.11 Effect of mean Base set on mean conflict. 92

A1.12 Effect of database size on response time. 92

A1.13 Effect of database size on I/O utilization. 93

A1.14 Effect of database size on CPU utilization. 93

A1.15 Effect of database size on mean throughput. 93

A1.16 Effect of database size on mean conflict. 94

A1.17 Effect of number of sites on mean response time. 94

A1.18 Effect of number of sites on I/O utilization. 94

A1.19 Effect of number of sites on CPU utilization. 95

A1.20 Effect of number of sites on mean throughput. 95

A1.21 Effect of number of sites on mean conflict. 95

A1.22 Effect of date rate on mean response time. 96

A1.23 Effect of date rate on I/O utilization. 96

A1.24 Effect of date rate on CPU utilization. 96

A1.25 Effect of date rate on mean throughput. 97

A1.26 Effect of date rate on mean conflict. 97

A1.27 Effect of date rate on network utilization. 97

A1.28 Effect of interarrival time on mean response time. 98

A1.29 Effect of interarrival time on I/O utilization. 98

A1.30 Effect of interarrival time on CPU utilization. 98

A1.31 Effect of interarrival time on mean throughput. 99

A1.32 Effect of interarrival time on mean conflict. 99

A1.33 Effect of interarrival time on network utilization. 99

A1.34 Effect of mean Base set on mean response time. 100

A1.35 Effect of mean Base set on I/O utilization. 100

A1.36 Effect of mean Base set on CPU utilization. 100

A1.37 Effect of mean Base set on mean throughput. 101

A1.38 Effect of mean Base set on mean conflict. 101

A1.39 Effect of database size on response time. 101

A1.40 Effect of database size on I/O utilization. 102

A1.41 Effect of database size on CPU utilization. 102

A1.42 Effect of database size on mean throughput. 102

A1.43 Effect of database size on mean conflict. 103

A1.44 Effect of number of sites on mean response time. 103

A1.45 Effect of number of sites on I/O utilization. 103

A1.46 Effect of number of sites on CPU utilization. 104

A1.47 Effect of number of sites on mean throughput. 104

A1.48 Effect of number of sites on mean conflict. 104

A1.49 Effect of date rate on mean response time. 105

A1.50 Effect of date rate on I/O utilization. 105

A1.51 Effect of date rate on CPU utilization. 105

A1.52 Effect of date rate on mean throughput. 106

A1.53 Effect of date rate on mean conflict. 106

A1.54 Effect of date rate on network utilization. 106

List of Figures

2.1 A deadlock situation in a database system. 13

3.1 Ethernet Architecture 23

3.2 Architecture of Token Ring 25

4.1 Overall Architecture 28

4.2 The schematic representation of states and events 30

4.3 Central site model for the centralized locking algorithm. 36

4.4 Other site model for the centralized locking algorithm. 39

4.5 Model for the distributed locking algorithm. 43

5.1 Effect of interarrival time on network utilization (%). 55

5.2 Effect of interarrival time on mean response time (sec). 56

5.3 Effect of interarrival time on mean throughput. 57

5.4 Effect of interarrival time on I/O utilization (%). 57

5.5 Effect of interarrival time on CPU utilization (%). 58

5.6 Effect of interarrival time on mean conflict (per transaction). 58

5.7 Effect of data rate on network utilization (%). 59

5.8 Effect of data rate on mean response time (sec). 60

5.9 Effect of data rate on mean throughput. 60

5.10 Effect of data rate on I/O utilization (%). 61

5.11 Effect of data rate on CPU utilization (%). 61

5.12 Effect of data rate on mean conflict (per transaction). 62

5.13 Effect of number of sites on mean response time (sec). 63

5.14 Effect of number of sites on mean throughput. 63

5.15 Effect of mean base set on mean response time (sec). 64

5.16 Effect of mean base set on mean throughput. 65

5.17 Effect of mean base set on conflict (per transaction). 65

5.18 Effect of database size on response time (sec). 66

5.19 Effect of database size on conflict (per transaction). 67

5.20 Effect of interarrival time on network utilization (%). 68

5.21 Effect of interarrival time on response time (sec). 69

5.22 Effect of interarrival time on mean throughput. 69

5.23 Effect of interarrival time on conflict (%). 70

5.24 Effect of data rate on network utilization (%). 70

5.25 Effect of data rate on response time (sec). 71

5.26 Effect of data rate on mean throughput. 72

5.27 Effect of data rate on conflict (%). 72

5.28 Effect of number of sites on response time (sec). 73

5.29 Effect of number of sites on mean throughput. 74

5.30 Effect of number of sites on conflict (%). 74

Chapter 1
Introduction

Distributed database management systems (DDBMS) [Ozs91, CeP84] have been the focus of intensive research during the past decade. Following the trend, we can easily foresee that their importance will rapidly grow. There are several technological and organizational reasons for this trend: (1) recent advances in microelectronics technology have made computer hardware affordable that previously was too expensive to be duplicated; (2) increased user demands for information have strengthened the need for faster information retrieval systems; (3) developments in computer networks have made computer communications cost effective and efficient; (4) advances in database technology have provided a solid foundation for the development of distributed databases; and (5) distributed databases seem to fit more naturally to today's large decentralized types of organizations [CeP84].

Since the field of distributed databases is relatively new, it is usually given different meanings and defined differently by different researches. Thus, in order to have a common frame for discussion, we will first attempt to define and explain what we mean by a distributed database. A *distributed database* (DDB) is defined as a collection of logically interrelated data items distributed over two or more computer sites interconnected by a network [Ozs91]. There are two elements of significance in the above definition. First, the sites are geographically separated and can communicate with each other only over some communication network. This is a necessary condition since by having a network as the only medium of communication another level of complexity is added to the problems of data distribution that otherwise might not be present. Second, data should have a certain amount of logical relationship which ties them together. This distinguishes a

distributed database from a set of local databases or files located at different sites of a computer network.

Concurrency control is concerned with the synchronization of concurrent accesses to a distributed database so that the consistency of the database is preserved. Since a distributed database system provides commands to read and write data that is stored at multiple sites of a network, they may interfere with each other by attempting to read and/or write the same data when users access a DDBMS concurrently. Concurrency control is the activity of preventing such interference.

Undoubtedly, the problems of concurrency control have received much attention during the last decade. Unfortunately, most of the work has concentrated on the development of new algorithms (e.g., [ELL77], [Lan78], [BaP78] and [Tho79]), one can name dozens of such algorithms that have been proposed, most of them with unproven workability and performance (in fact, it has been claimed that many of them are incorrect [BeG81]). Most distributed concurrency control methods fall into one of three basic classes: locking methods, timestamp methods and optimistic methods. A survey and an analysis of these algorithms can be found in [BeG81]. Which concurrency control method performs best depends on many characteristics of the distributed database, how it is used, and the computer systems and communication network upon which it is based. Some researchers have already taken first steps towards this goal, but there is still much work to be done. The issue is important because the concurrency control module can easily become the performance bottleneck in a DDBMS.

There are many performance studies of concurrency control algorithms (e.g., [Gar79], [Rie79], [Che81], [LiN83], [Ozs85b] and [AhB88]). These typically measure the throughput capacity that can be supported by the method, the average response time of transaction, and the total amount of system resource (CPU time, communication

bandwidth, etc.) used by the algorithm (i.e., the overhead). Generally, these performance measures are determined by a number of input parameters, such as number of sites, number of items at each site, number of transaction classes and multiprogramming level of active transactions at each site etc. They typically assume a simplified network model, however, and that is the emphasis of this thesis.

1.1. Objective of Thesis

The existence of a communication network is an integral part of a distributed database system, yet the effects of the network on the performance of distributed database systems have received little attention in the literature. In most cases the effect of the network is reduced to a certain time delay assumed to be constant under different load conditions and various transaction types.

In some situations such an assumption could lead to very misleading performance results. As an example, consider the response time of a distributed database system with an Ethernet [MeB76] (for a description of which, see Chapter 3) as the underlying network architecture. Under very heavy load conditions, the response time of an Ethernet is unbounded, i.e. not finite. Substituting a constant time value for network delay in such a case obviously is not realistic.

The primary objective of this thesis is to examine how the performance of distributed database concurrency control algorithms are affected by the underlying network architecture. Specifically, we consider two algorithms: *centralized locking* [Gar79] and *distributed locking* [GaC80] algorithms.

Here, we are concerned with Local Area Networks (LANs). A local area network is distinguished from other types of networks in that it is usually confined to a moderate-size geographic area (usually less than 10km), has higher speed (1-100 Mbytes per

second-Mbps), and a very low error rate. Local area networks usually have well defined, regular structures, e.g. a ring or a star. Moreover, a detailed performance analysis has been performed on most LAN topologies.

In this thesis, the performance of two concurrency control algorithms is studied in two different network environments.

The objective of this study are two-fold:

(1). To better understand the behavior of these two algorithms. Previous performance analyses of these algorithms [Ozs85a, Ozs85b] have assumed a constant transmission time. The current study should provide a more realistic comparison.

(2). To understand the impact of the network protocols on the performance of concurrency control algorithms. This analysis has not been carried out before, and should provide some guidelines for the design and implementation of distributed database concurrency control managers. The two network protocols that we consider are the token ring and the CSMA/CD bus - more specifically Ethernet.

1.2. Organization of Thesis

The thesis is organized as follows. In Chapter 2, we present a review of the background material on distributed databases and introduce the basic techniques that are used for concurrency control. In Chapter 3, we include a discussion of some distinct features of LANs. In Chapter 4, the simulation models are introduced. The results and analysis of the simulation experiments are provided in Chapter 5. We investigated the effects of LANs on DDB systems in general in Chapter 6. Finally, in Chapter 7, we give the conclusions of the study, and make some of suggestions for further research.

Chapter 2
Concurrency Control in Distributed Database Systems

This chapter introduces some of the problems that are commonly encountered in designing distributed database systems and surveys several of the techniques that are commonly used to handle these problems. Among the topics that are discussed are concurrency control and deadlock resolution. At the same time, the database model for this study is presented.

2.1. Terminology

A distributed database (DDB) is defined as a collection of logically interrelated data items distributed over two or more computer sites interconnected by a network. In a distributed database, each data item may be stored at any site in the system or stored redundantly at several sites. A distributed database is said to be *partitioned* if there are no duplicate items; *partially replicated* if part of the data is duplicated; and *fully replicated* if the entire database is duplicated at all sites. In this study, we are concerned mainly with replicated databases.

A distributed database is said to be consistent if all replicated portions of the database are both internally consistent and mutually consistent. *Internal consistency*, which is fundamental to both centralized and distributed database systems, implies that the entity values in any copy of the database satisfy a set of integrity constraints. Integrity constraints refer to *correctness assertions* that are associated with the values of every data item in the database. An example of such assertions is: "The salaries of employees in department x should be between 2,000 dollars and 80,000 dollars". *Mutual consistency*, which is specific to distributed database systems, means that all the values of

multiple copies of any data item converge to the same final value should the system stop receiving new transactions.

A user interacts with a database by means of *transactions*. A transaction is defined as a sequence of primitive atomic operations (for example, reads, computes and writes) that maps the database from a consistent state to another consistent state [Lam78]. A transaction is thus a larger unit of atomic action on the database state.

Transactions could be categorized into two types: *read-only* transactions and *update* transactions. In this thesis, we are interested mainly in update transactions. An update transaction is assumed to consist of the following steps:

1. **Read** - the items needed by the transaction are read.
2. **Compute** - new values are computed.
3. **Write** - the distributed database is updated, that is, all duplicate items that need to be modified are updated to reflect the new values.

A transaction must possess the following properties:

Atomicity: Either all or none of the transaction's operations are performed. Thus once a transaction begins, it must either be *committed* as a whole or *aborted*. Atomic commitment is problematic in distributed database systems. Suppose a transaction is updating a data item at a number of sites, and while in the commitment phase and after updating the data item at some sites, but not all, its transaction manager fails. If not properly handled, this will result in an inconsistent database. Moreover, other transactions will access the inconsistent database.

Durability: Once a transaction has committed, its results must be stable. In other words, the results produced by the transaction should never be lost even in the face of failure.

Serializability: An execution is serializable if it is computationally equivalent to a serial execution, that is, if it produces the same output and has the same effect on the database as some serial execution [BeG81]. This is important to maintain intra-consistency between transactions in a DBMS. Serializability requires that there be a total order of execution of conflicting transactions. Two transactions are said to conflict if they operate on the same data item and one of them is a write. The activity of guaranteeing serializability is called concurrency control.

Durability can be ensured via database recovery techniques, and is not addressed in this thesis. Serializability will be discussed in the following section.

2.2. Distributed Database Consistency Problems

In a distributed database, there are basically two types of problems: (1) the internal consistency problem, and (2) the mutual consistency problem.

Internal consistency problem: The internal consistency problem arises when several conflicting transactions attempt to modify a copy of the database at the same time. This situation is illustrated by the following example. Consider only one copy of a distributed database and the following two conflicting transactions:

```
T1: Read x;              T2: Read x;
    x1:= x + a;              x2:= x - b;
    Write x1 into x;         Write x2 into x;
```

If the two transactions are executed concurrently, it is possible for the transactions to read the data item x, compute the new value for x, and then store the new value into the database *at approximately the same time*, as shown below:

```
T1: Read x;
T1: x1:= x + a;
T2: Read X;
T2: x2:= x - b;
T1: Write x1 into x;
T2: Write x2 into x;
```

If this happens, the final value of x is incorrect, since the effect of one transaction is overwritten by the other transaction. In the example above, the effect of T1 on x is lost and the final value of x is $x-b$ instead of $x+a-b$ or $x-b+a$, as one would expect. This is known as the lost *update anomaly* and is one of the several types of internal inconsistencies [BeG81] that could occur in both the centralized and distributed database systems. Update anomaly points to the necessity of synchronizing the concurrent execution of conflicting transactions. Two transactions are conflicting if they operate on the same data item and one of them is a write.

Mutual Consistency Problem: The mutual consistency problem occurs in fully or partially replicated databases because of the requirement that copies of the database must be identical. In order to keep all copies identical each update transaction must be applied uniformly and simultaneously to every copy of the database. However, because of communication delays and failures, it could happen that updates are applied to different copies at different times and in different orders. If this occurs and the updates are not controlled, the mutual consistency of the database could be affected. This situation is illustrated in the following example.

Consider two sites A and B, geographically separated and linked together by a network, each one with a duplicate copy of the database. Assume that two transactions are received by site A and that the transactions are similar to T1 and T2 above except that before a new value is written into a copy of the database, the value is sent to the other site

to be stored in the other copy as well. Now, although we assume that an internal consistency check is performed and T1 and T2 are executed sequentially to prevent lost updates, the consistency of the distributed database is still not guaranteed. Inconsistency could occur if the messages sent by one site are not received and processed in the same order by the other site. Note that in a distributed system, there is no guarantee that the messages will be received in the same order that were sent. This is illustrated below:

Site A	Site B
T1: Read x;	Receive x2 from site A;
x1:= x + a;	Write x2 into x;
Send x1 to site B;	Receive x1 from site A;
Write x1 into x;	Write x1 into x;
T2: Read x;	
x2:= x - a;	
Send x2 to site B;	
Write x2 into x;	
Final value of x: x + a - b	Final value of x: x + a

When this occurs the final state of the database is inconsistent since the transactions are not executed in the same order at the two sites and the value of a more recent update is overwritten by an older update. In the above example, the final value of x at site B is $x+a$ instead of $x+a-b$ which is the final value of x at site A.

2.3. Internal Consistency Mechanisms

One obvious solution to the internal consistency problem is to run the transactions serially, one at a time in any order. Since a transaction is a unit of consistency, any sequence of transactions executed serially without interference from other transactions also preserves consistency of the database. However, this is not a good solution since there are transactions, for example, those that do not access the same data items, that can be executed in parallel or concurrently without affecting the database consistency.

Therefore, what would be more appropriate is a mechanism that would allow transactions to be executed concurrently without violating the consistency of the database. In formal terms, the mechanism would ensure that the execution of concurrent transactions is *serializable* ([BeG81], [Pap79]); that is, even though the transactions are executed concurrently the overall effect on the database is equivalent to what would have resulted if the transactions were executed in some serial order. Such a mechanism is provided by concurrency control algorithms. Many algorithms have been proposed in the literature, and the mechanism that is almost exclusively used in commercial systems is *two-phase locking*. In this thesis, we are concerned only with two-phase locking algorithms as well.

Two-phase locking achieves serializability by using locks to isolate conflicting transactions from each other and by requiring that locking and unlocking of items be done in two phases, known as the *growing phase* and the *shrinking phase*. During the growing phase the transaction can only request locks, and during the shrinking phase the transaction can only release locks and cannot request any additional locks. It has formally been proven that two-phase locking is a correct concurrency control method [CeP84].

2.4. Mutual Consistency Mechanisms

The problem of maintaining the mutual consistency of distributed databases has also received a lot of attention in the past few years, and an abundance of algorithms exists in the literature. One common solution is to modify the 2-phase locking mechanism so that it preserves the mutual consistency of the distributed databases as well. The two basic strategies that are usually incorporated with the mechanism are known as *centralized locking* and *distributed locking*. The two algorithms studied in this thesis are actual implementations of these two strategies.

2.4.1. Centralized Locking

In this strategy, one of the sites, called the *central site,* is chosen a *priori* (using an election protocol [Gar79], [see Section 4.4.1 also]) to control the allocation and deallocation of locks to transactions. Before a transaction can access data at any site the appropriate locks must be requested and obtained from the central site. For example, if a transaction wants to update item x, the site where the transaction originates must send a lock request for x to the central site, wait for the lock granted message from the central site, and then proceed with the update of item x. After the update is completed at all the sites, a message to unlock the item is sent to the central site. This mechanism ensures a total ordering among conflicting transactions so that both internal and mutual consistencies are preserved.

The common criticisms against this strategy are: (1) the central site could be a performance bottleneck since all the update transactions must visit the site to obtain the locks, and (2) the reliability of the entire system is too dependent on one site: a central site failure causes a total system failure. Some approaches to improve the reliability of the scheme have been suggested. Alsberg and Day [AID76] have shown that by keeping a backup of the central site, any desirable level of reliability can be achieved. However Cheng [Che81] has carried out performance studies and has found that the performance of the resulting scheme is affected considerably due to the additional overhead incurred. Garcia-Molina [Gar79] has suggested an alternative approach that uses election protocols to rapidly recover from a central site failure, but no studies have yet been carried out to find how this scheme would perform. In this thesis, we study this scheme and we draw some conclusions regarding its performance.

2.4.2. Distributed Locking

Instead of having the lock management duties performed at only one site, these duties could be performed at every site in the system. However, in this strategy, before a transaction could update a data item at any site, locks for the items must be requested and obtained from every site in the system. Similarly, after the completion of the update at every site, the locks for the item should be released at every site. The major difference between this strategy and the centralized locking strategy is that, in this strategy, a lock table is kept at every site so that: (1) the read-write or write-read conflicts between transactions could be controlled locally, and, as a result, only write locks need to be requested from the other sites; and (2) failure of any site does not cause a total system failure since the lock table could still be accessed from any operational site.

The distributed locking strategy has been criticized mainly because of the complexity of the algorithm and the extra overhead that is incurred during normal operation.

2.5. Deadlock

One problem with locking-based mechanisms is that they are subject to deadlock and therefore need deadlock resolution mechanisms. A *deadlock* is said to have occurred in a database system when a transaction T_i is holding a lock for an item d_i and is waiting for an item d_j locked by a transaction T_j which is directly or indirectly waiting for item d_i. A simple example of a deadlock is illustrated in Figure 2.1. An arrow from a transaction, depicted as a rectangle, to an item, shown as a circle, indicates that the transaction is waiting for the item. An arrow from an item to a transaction indicates that the item is locked by the transaction.

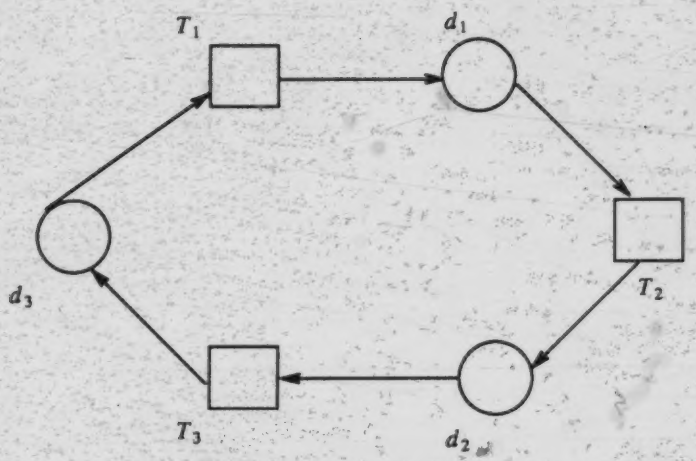


Figure 2.1 A deadlock situation in a database system.

There are three schemes that are commonly used for handling deadlock. These include *deadlock prevention*, *deadlock avoidance*, and *deadlock detection*.

The first two schemes are self-explanatory. The most complicated step is the last, namely, how to detect deadlock?

In the deadlock prevention scheme, if a transaction T_1 requests a resource that is already held by another transaction T_2, a deadlock "prevention test" is run. If the test indicates that there is a possibility of deadlock, either T_1 is canceled (non-preemptive) or T_2 is aborted *(preemptive)*

In the deadlock avoidance scheme, transactions are required to request their resources in some predefined order, and a transaction can only wait for an item which is held by an older transaction (or one with a higher priority). In this scheme deadlocks cannot occur. The algorithms studied in this thesis are based on this scheme.

In the deadlock detection scheme, deadlocks are detected and resolved after they occur. Two mechanisms could be used to detect a deadlock: *timeout* and *Wait-For-Graph* (WFG). If the timeout mechanism is used, a deadlock is assumed to have

occurred if after a timeout period a transaction is still waiting for an item. When this situation occurs, the transaction is aborted. This mechanism can cause unnecessary aborts. A WFG depicts the waiting sequence of transactions for access to a data item. If the WFG mechanism is used, deadlocks are detected by searching for cycles in the graph and then resolved by aborting one or more of the transactions involved to break the cycles.

In a centralized database system, deadlocks could easily be detected since all the information about the WFG is located at one place. However, in a distributed database system, deadlock detection is much harder since the information is dispersed among the sites and, furthermore, deadlocks can occur not only locally (that is, involving only one site) but also globally (that is, involving more than one site). Detection and resolution of deadlocks in distributed systems are discussed in [Mal80] and [Obe82].

Chapter 3
Distributed Databases in a LAN Environment

In this chapter, we briefly discuss some distinct features of LAN that have a direct influence on some of the design aspects of a distributed database system.

3.1. LANs and their Distinct Features

A *local area network* (LAN) is a communications network that provides interconnection of a variety of data communicating devices within a limited geographic area. When compared to a *wide area network* (WAN), LANs possess certain distinct features that have a direct influence on some of the design aspects of a DDB. These include:

(a) very high data rates

The data rates of currently operational LANs could reach several million bits per second (for example, Ethernet [Dec80] operating at 10 Mbps) while LANs with data rates reaching 100 Mbps (FDDI ring [Ros86]) or over are being designed for future requirements.

(b) Limited distances spanned

A typical LAN may span a few kilometers. Therefore, the medium propagation delay between two sites connected by the same LAN is usually very low. For applications requiring communication over longer distances, LANs may have gateways to a WAN. In addition, similar LANs may be interconnected using bridges. Therefore, at application level, generally there is no distance restriction on communication over interconnected LANs.

(c) Low error rates

Typically, LANs will have lower error rates (10^{-8} to 10^{-11}) than WANs. Therefore, the time spent in message retransmissions will be much less. This results in an overall improvement in the response time of an application.

(d) Broadcast capability

In widely used LAN topologies such as bus and ring, a message transmitted by one site reaches all other sites connected to the medium. Using the destination address contained in the message, only the addressed destination will receive it while other sites either ignore or discard it. By using an address known to all sites (a broadcast address) or an address known to a group of sites (a multicast address), a message can be broadcasted to all or some selected sites. In contrast to WANs, the cost of sending a message to a single site or multiple sites will be the same in most LAN architectures. Therefore, in cost models where transmission cost is taken into consideration, the ordering of sites and distances among them need not be considered in a message broadcasting environment.

Even though there are other differences between local and wide area networks, the above include the most important features that influence the design and implementation of DDBs in a LAN environment.

3.2. Influence of LAN features on DDB Design

In this section, we will discuss how the design aspect of distributed databases could be influenced by LAN features in general. Further, we will see how these features could be used to achieve an overall improvement in the performance of a DDB system.

3.2.1. Data Fragmentation and Allocation

In a DDB, a major reason for data fragmentation is to increase the level of concurrency. While achieving this objective, the fragmentation could introduce several other problems [Ozs91] such as:

(a) possible degradation in view management performance when views are over several distributed fragments

(b) possibility of expensive operations such as joins in query processing

(c) difficulties in integrity enforcement.

In a LAN environment, it could be possible to achieve the desired level of concurrency with less or no data fragmentation. This is because, the lower cost of transmission and lower transmission delays in a LAN allows one to consider despatching even whole relations to preferred sites for concurrent processing, where cheaper processing power or more processing capabilities are available.

Considering the allocation of fragmented data, either single or replicated copies could be maintained over the distributed sites. Reasons for data replication are the enhanced reliability and the improvement in efficiency in processing read-only queries [Ozs91]. However, data replication results in increased storage requirements. The trend of decreasing storage costs with technological developments could continue and therefore storage costs may cease to be a major cost factor. A major drawback in data replication is the complexities introduced in updating replicated copies. It is essential to ensure that all copies are correctly and more or less simultaneously updated to maintain the consistency and integrity of the database.

The overall reliability of a LAN is expected to be higher than that of a WAN due to the technical and administrative factors such as:

(a) much less (or no) routing or switching media

(b) more homogeneous transmission media

(c) less complicated and sophisticated hardware interfaces and software protocols

(d) Ownership and maintenance by a single organization

(e) ease of fault detection, isolation and repair due to limited geographic areas involved.

As communication delays are lower in a LAN, even with less or no replication of data, it could be possible to achieve a reasonable response time in read-only queries. Lower communication costs make retrieving data from distant sites over the network economically feasible.

The above discussion suggests that data fragmentation and allocation issues have to be carefully reviewed within a LAN framework. The advantages gained by data fragmentation and replication over a LAN could be much less than in a WAN. Further, in a LAN environment, the advantages gained by data fragmentation or replication may be offset by the complexities introduced by them. As communication delays are lower in a LAN, even with less or no replication of data, it could be possible to achieve a reasonable response time in read-only queries. Lower communication costs make retrieving data from distant sites over the network economically feasible.

3.2.2. Distributed Query Processing

The processing of a query in a DDB involves the identification of the set of physical fragments referenced by the query, determining an order of execution of operations and selecting the method for executing each operation. In most of the studies done on distributed query processing, it is assumed that the transmission cost per unit data is constant for any two points in the network and the cost of local processing is negligible when

compared to the transmission costs. In optimization strategies, total transmission cost, overall response time or both have been considered as the objective function to be minimized. In instances where total transmission cost has been used as the objective function, the assumption has been that the cost of transmission is the dominant factor. The validity of this assumption becomes questionable when the underlying network is a LAN. Further, usually static strategies have been used due to the assumed high transmission costs in collecting database statistics required for dynamic optimization strategies. Within a LAN environment, with lower transmission delays and costs, possible advantages that could be gained by using dynamic strategies have to be considered.

In [WaY85] the effective use of the broadcast property of a local network in dynamic query processing is discussed. The study considers query processing and concurrency control in an integrated fashion within the context of a broadcast bus network. In [HeY87], distributed query optimization algorithms for token ring and broadcast type networks are presented. These algorithms take into account both data transmission and local processing costs. The algorithms adopt a heuristic and static optimization approach.

Hevner and Yao [HeY87] have presented a survey on distributed query optimization for local area networks. They have pointed out that the potential for dynamic optimization techniques in local networks is greater than in point-to-point networks. The reason is that the system state information can be transmitted to all sites more rapidly in a LAN. The contention for network resources (for example, waiting time before gaining access to the network itself) is recognized as an additional cost factor that should be included in the query optimization cost models. However, it would be very difficult to estimate the contention costs accurately as it requires detailed system information at query execution time.

3.2.3. DDB Administration Issues

In this section we will briefly consider effective solutions for DDB administrative issues such as catalog management and user authorization mechanisms within a LAN framework.

In a DDB, catalogs contain information required for accessing the database. This includes the description of data fragmentation and allocation and the mappings to local names [CeP84]. User authorization is the process of identifying legal users of the system to ensure database security. Another aspect of security is the enforcement of authorization rules to regulate the actions performed on database objects. In a DDB, several alternative strategies are available for catalog management and user authorization. For example, a complete catalog could be kept at a central site, catalogs could be fully replicated or they could be fragmented and allocated to local sites. In user identification, in principle, it should be possible for users to identify themselves at any site of the system [CeP84]. This could be achieved either by replicating passwords at every site or allocating "home sites" for users, where identification is performed. The trade-offs between these different strategies are discussed in [CeP84].

Depending on the degree of local site autonomy desired, an effective solution within a LAN configuration could be to have a "server" on the network for the management of catalog and authorization information. Lower communication delays and costs make this a practical solution for a DDB implementation over a local network. Obvious advantages of this approach would be the ease of updating the information and the less storage required. However, the reliability of the serveer should be guaranteed by duplicating it or otherwise, without significantly increasing the overall cost and the complexity of the system. Various protocols (for example, TCP/IP) include explicit support for such servers.

In the above sections, we have discussed how some of the DDB design issues could be affected within a LAN setting in general. In the following sections, we will specifically concentrate on how the transaction management in a DDB could be affected by the distinct architecture and protocol features of Ethernet and Token Ring local area networks.

3.3. Ethernet and Token Ring Local Area Networks

In our simulation experiments we have selected Ethernet and Token Ring as the networks over which concurrency control algorithms are implemented. The following sections describe the justification for choosing these networks, a brief description of their architectural and protocol features and ideas on how these features could affect the performance of transaction management in a DDB.

3.3.1. Why Ethernet and Token Ring?

Primarily we were motivated by the following factors in specifically concentrating our study on the Ethernet and Token Ring networks.

(a) They use very common LAN topologies.

All LAN schemes known to us use bus, ring, star, tree or minor variations or combinations of these as their topology. Out of these, most of the LANs use either a bus or a ring topology. Ethernet uses a bus architecture while the token ring consists of nodes connected in a ring fashion. Therefore, a study based on these networks could reveal any specific advantages or disadvantages on the overall performance caused by these common LAN architectures. An important observation here is that the performance could mainly be affected by the protocol features of the LAN and therefore the architectural features could be of secondary importance. Therefore, in the interpretation of performance results, the combined protocol and architectural features should be considered

rather than considering them in isolation.

(b) They use standardized protocols.

Ethernet uses the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol specified by the IEEE802.3 standard and the token ring protocol is standardized by the IEEE802.5. As a result of these standardizations, commercial LANs using these protocols are widely available. As already observed, the protocol features of a network are of primary importance in the performance study of a DDB. Therefore, our study could reveal specific features of these protocols that could be extended to other networks using the same protocols.

(c) They are commercially available.

Ethernet is one of the local area networks that has been implemented on a large scale. On the other hand, IBM Token Ring is an example of a commercially available token ring conforming to the IEEE 802.5 standard. It is quite possible that distributed databases will be implemented over these networks in developing integrated office information systems or other application environments. Therefore, our study based on these networks could reveal important considerations that are of practical significance.

3.3.2. Ethernet and the CSMA/CD Protocol

The original Ethernet [MeB76] was developed and implemented on an experimental basis in 1975. Later in 1980, specification for a revised Ethernet was released jointly by Dec, Intel and Xerox Corporations.

Ethernet uses a shielded coaxial cable as its medium with a broadcast bus topology (Figure 3.1). The original Ethernet was designed for a data rate of 3 Mbps while the revised Ethernet specification provides for an enhanced rate of 10 Mbps.

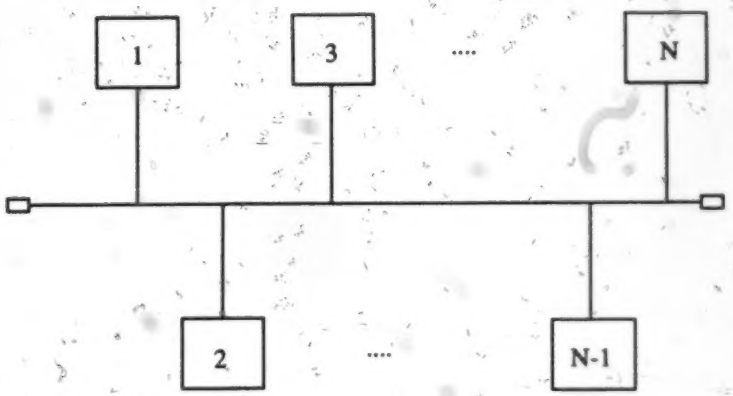


Figure 3.1 Ethernet Architecture

Ethernet uses the CSMA/CD protocol [Ans85] to provide access to the transmission medium by multiple users. In this access mechanism, a station wishing to transmit first listens to the medium (channel). If the channel is sensed busy, it defers to the ongoing transmission and waits. When the medium is ultimately sensed free, the station begins its transmission. It is possible that two or more stations sense the channel idle and begin their transmissions more or less simultaneously, producing a collision. Therefore, each sender continues to monitor the channel during transmission to detect any collisions. If a collision is detected, the transmission is continued for a brief interval (called a "jam") to ensure that the collision is heard by all other transmitting stations. Then the station stops its transmission and schedules a retransmission attempt after some delay. The range of this delay is decided according to a 'truncated binary exponential back-off algorithm' to avoid repeated collisions.

The retransmission delay is an integral multiple of 512 bit times (known as the "slot time"). The number of "slot times" to delay before the n^{th} retransmission attempt is chosen as a uniformly distributed random integer r in the range $0 \le r \le 2^k$ where $k = min(n, 10)$. This means that the interval from which the retransmission time is decided

is expanded exponentially with repeated collisions (up to some limit). Therefore, a packet that has already experienced several collisions could suffer a longer delay than a packet that has experienced fewer collisions.

The proper operation of the CSMA/CD scheme requires that all transmitting stations agree as to whether a collision has occurred. To ensure this, Ethernet specifies a minimum packet length of 64 bytes (46 bytes of data). The round- trip propagation delay is about the same amount of time it takes to transmit 64 bytes at 10 Mbps over a maximum-sized Ethernet configuration. Therefore, a transmitting station will, even in the worst case, still be transmitting the same packet when it hears a collision.

Ethernet provides minimum delay at light loads because of the random access nature of the CSMA/CD protocol. However, under heavy loads, collisions occur due to simultaneous transmission attempts by several stations. These collisions and the resulting back-off delays make the packet delay in Ethernet unbounded at heavy load conditions.

3.3.3. Token Ring Architecture and Protocol

The token ring consists of a set of stations serially connected by a transmission medium to form a closed loop as shown in Figure 3.2. The version of the token ring being adopted by the IEEE 802.5 is an outgrowth of research and development at the IBM. The data rate used is 4Mbps.

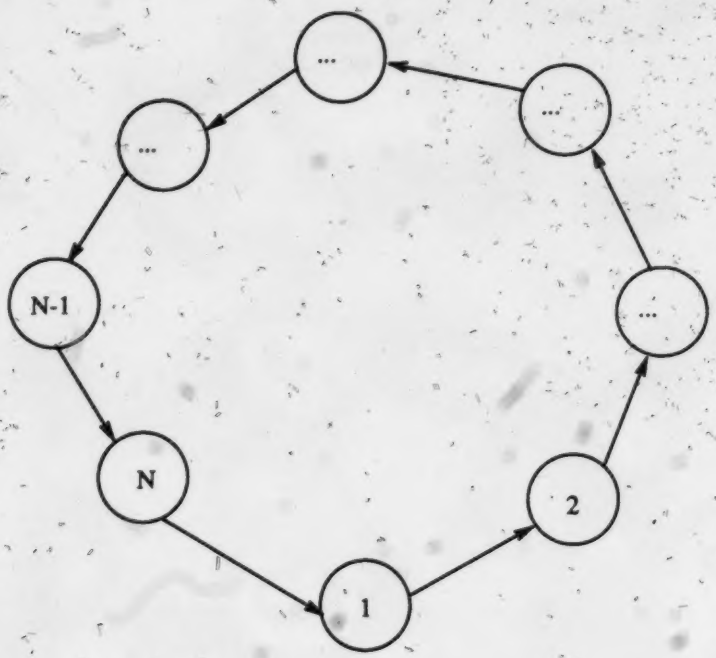


Figure 3.2 Architecture of Token Ring

Access to the medium is controlled by the use of a token (a uniquely identifiable bit pattern) that circulates round the ring. A station wishing to transmit has to wait for a free token to pass by. It then converts it to a busy token and start its transmission. Every active station in the ring will regenerate and repeat each bit it receives. The addressed destination will copy the contents of the message. Ultimately when the message returns to the sender after one rotation in the ring, the sender converts it into a free token. Therefore, under normal operation, access rights to the medium are passed from one station to the next in a round-robin fashion. A token holding timer prevents a station from indefinitely holding the token and hogging the medium.

The controlled access to the medium provided by this mechanism makes the maximum delay that a station may have to experience predictable. Therefore, even under heavy load conditions, a fair response time can be expected from token ring networks.

However, a station has to wait for a free token to arrive to start its transmission. Therefore, there could be an initial waiting time (on average, half the ring propagation delay) even if all other stations are idle. This is generally viewed as a drawback in token passing access schemes.

Possible problems in a token ring include:

(a) A station failure, depending on its nature, may cause a total network failure. This type of failures are possible as the station interfaces are active in a ring network. Therefore, suitable station bypass mechanisms and network reconfiguration techniques have to be employed to enhance the network reliability and availability.

(b) Packets with damaged addresses may circulate indefinitely.

(c) A sender not introducing a new free token could result in the disappearance of the token from the ring. A free token not converted to busy at the beginning of transmission could end up in having multiple tokens in the ring. Therefore, one or more stations should be capable of performing ring monitoring and initialization functions.

Chapter 4
The Simulation Model

In this chapter, two issues are covered; (1) the concurrency control algorithms that are studied in this thesis are presented, and (2) the local area network models are discussed.

4.1. Overview of The Simulation Model

As mentioned before, the primary objective of this thesis is to examine how the performance of distributed concurrency control algorithms are affected by the underlying network architecture. Logically, the entire simulation system could be viewed as consisting of two distinct subsystems: the concurrency control subsystem, and the local area network subsystem. The overall architecture is depicted in Figure 4.1.

The simulation system presented in this thesis is developed on the LANSF (Local Area Network Simulation Facility) simulator package developed by Gburzynski and Rudnicki [Gbu90]. LANSF is configurable simulator dedicated to model communication networks. The attributes of a physical system that can be specified in LANSF are divided into two categories. The first category contains static elements, i.e. the system architecture and topology. The second category consists of the dynamic attributes that describe and mirror the temporal behavior of the modeled system, i.e. the traffic patterns, the communication protocol, the performance measures.

Figure 4.1 Overall Architecture .

LANSF uses the concept of a message as a complete information unit. A message is broken up into several packets, as necessary, and transmitted to the receiver. LANSF can be used to model any type of a physical system in which communication is the most critical issue. In particular, distributed computer architectures (their communication aspects) can be naturally described in LANSF. LANSF has already been used to model a multiprocessor computing system used to run a distributed simulator.

4.2. Modeling of The Concurrency Control Algorithms

The concurrency control subsystem can be modeled as a collection of local schedulers $\{LS_n\}$ such that LS_n is the local scheduler running at site n. The local schedulers represent the distributed components of the overall concurrency control protocol in a distributed database.

Conceptually, a LANSF implementation can be thought of as a series of state transitions. A process in a certain state changes to a new state when a specific event occurs. It can be said that a process waiting for a specific event is "sleeping" and when the event occurs, any process which was waiting for that event is "waken up". So we model each local scheduler as a finite state machine, with independent state and event specifications as well as local data structures. All local schedulers are not necessarily homogeneous. For example, a synchronization protocol with centralized control may have one local scheduler running on a central controlling site and all other local schedulers as the slave processes. In such a case, the finite state machine for the central scheduler is more complex.

Each local scheduler LS_n can be characterized by a quintuple $<S_n, E_n, M_n, D_n, L_n>$ where S_n is a set of states, E_n is a set of events, M_n is a set of message types, D_n is a set of local data structures, and L_n is a set of local operations on M_n and D_n.

A state of a local scheduler represents a finite period of local computation. In a state, a protocol can perform any number of operations with one exception: sending and receiving messages must be modeled as state transitions. Examples of legal operations in a state include: operations on local data structures, message preparations, predicate evaluation over local data and/or message contents, etc.

An event is an activity that causes a scheduler to change its state. For example, as

required by this model, sending and receiving messages constitute events. An event involving no message exchange models a transition between processing stages of a protocol. In this case, the event signifies the completion of the processing in the previous state and start of the next state. For example, in Figure 4.2, event e causes the scheduler to change from state S_1 to S_2. Message *MESS* is sent when event f takes place causing a change from state S_3 to state S_4.

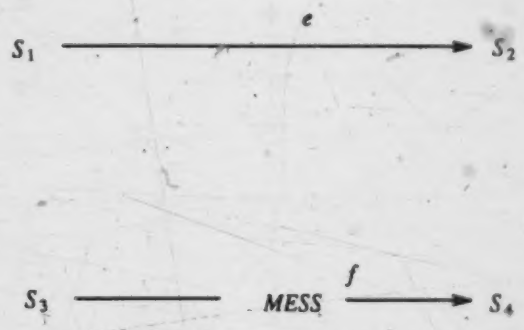


Figure 4.2 The schematic representation of states and events

The message types M_a specify message id's and message formats. The local data structures D_a contain all local variables accessible only by LS_a. And the local operations of LS_a, L_a are defined over M_a and D_a. Computational details are specified in L_a.

4.2.1. Centralized Locking Algorithm

One of the algorithms that we study is the Centralized Locking Algorithm with Hole Lists (MCLA-h) proposed in [Gar79]. The MCLA-h is a predeclaration strict two-phase locking-based algorithm. Thus, transactions are not allowed to execute before acquiring all locks. Moreover, locks are not relased before the execution of the transaction is completed at its original site. Deadlock prevention is implemented by simply predeclaring and obtaining all locks.

In this algorithm, the database is fully duplicated and is viewed as a collection of

named items (tuples). Only update transactions are considered, i.e., all locks are exclusive. Because the database is fully duplicated, there is no the need to consider the effects of directory management.

One very important reason that motivated us to use it in this study is that this particular algorithm is frequently referred to as being an example of deadlock prevention 2PL algorithms. Moreover, the deadlock prevention technique used is somewhat easier to implement: no conflicting transactions are allowed to execute at the same time. From performance point of view, we wanted an algorithm that can make use of the broadcast property of the Ethernet and Token Ring and at the same time would provide contention on the network. The MCLA-h algorithm that we implemented does just that in the "perform update" and multiple simultaneous "lock requests" respectively.

A summary of the MCLA-h algorithm is:

Step 1: An update transaction A arrives at node x from a user.

Step 2: Node x requests from the central node (predetermined) locks for data items referenced by the transaction.

Step 3: The central node checks for all of the requested locks. If all locks can be granted, a "grant" message is sent to node x. If, however, some items are already locked, then the request is queued. There is a queue for each data item and a request only waits in one queue at a time. All transactions request locks for their items in the same predefined order to prevent deadlocks.

Step 4: The central node provides a sequence number to all transactions it grants locks to.

Step 5: The central node maintains a list of transactions with locks granted, called the "hole list", $H(A)$. When the locks of an update transaction are granted, the

transaction's sequence number is added to the hole list. This list is sent to node x when its requested locks are granted.

Step 6: When transaction A originating at node x is granted its locks by the central node, it must wait until all transactions with lower sequence number than A's sequence number, but are not in the hole list $H(A)$, have completed at node x. The data items requested by A are then read and values are computed. A "perform update" message containing the hole list and transaction sequence number $S(A)$ appended to it is sent to all nodes. Before another node performs the update on behalf of this transaction A, it must ensure that it has already performed all updates with lower sequence number than $S(A)$, which are not in $H(A)$.

Step 7: When the central node receives a "perform update" message, it releases the locks associated with the corresponding transaction and removes the transaction sequence number from the hole list. Other requests waiting on the released data items are then processed.

4.2.1.1. Central Site Model

In modeling this algorithm, the central site model should be distinguished from the model of the other sites since the local operations performed are quite different.

S_c: states = $\{S_0, S_1, S_2, \ldots, S_{12}\}$

E_c: events = $\{e_0, e_1, e_2, \ldots, e_{21}\}$

M_c: message types = $\{MESS_1, MESS_2\}$

Here, messages of type $MESS_1$ can be sent only to one destination site and messages of type $MESS_2$ can be sent to all the sites.

The local data structures are presented as follows:

D_c: local data structures = {

d_i: data item i; i = 1, 2, ..., database size;

sq_no: sequence number;

h_list: hole_list in system;

dq_i: queue for transactions which conflict with each other due to locking d_i;

U_b: the set of blocked transactions;

q_0: queue for pending transactions, such as new local update transactions and transactions from other sites which have lock requests;

q_1: queue for ready updating transactions;

T: transaction currently being executed;

base_set(T): base_set of transaction T;

$H(T)$: the hole list of transaction T;

$S(T)$: the sequence number of transaction T }

The local local operations are presented as follows:

L_c: local operations = {

enqueue(q, X): put X into queue q;

dequeue(q, X): remove X from queue q;

enset(U, X): put X into set U;

deset(U, X): remove X from set U;

Here, X is an object of any kind of data structure.

read(T): read items from local database required by transaction T from *read_set*(T);

compute(T): compute new value for transaction T;

write(T): write new value for transaction T into local database;

check-hole(T): check if all transactions with lower sequence number than T's sequence number, but that are not in the $H(T)$, have completed at this site.

check-empty(X): check if queue X or set X is empty;

check-local(T): check if transaction T is local;

check-blocked(T, T_b): check if there is a transaction in U_b blocked by T, if yes, T_b = blocked transaction, if not T_b = empty;

check-dataitem(d_i): check if data item d_i is locked;

lock-dataitem(d_i): lock data item d_i;

release-dataitem(d_i): release data item d_i;

sorting(*base_set*(T)): arranging base set of T in the ascending order (for avoiding deadlock);

prepa_mes(*Mess*, T): preparing message of *Mess* type according to T;

commit(T): T commits;

copy(X, Y): copy X to Y; }

Now we need to specify the states and events in terms of the local operations.

State: (in state S_i, the local operations that takes place are)

S_0: idle;

S_1: enqueue(q_0, T); sorting(*base_set*(T));

S_2: enqueue(q_1, T);

S_3: check-empty(q_0); dequeue(q_0, T); loop: d := min(*base_set*(T)); deset(*base_set*(T), d); check-dataitem(d_i); lock-dataitem(d_i); until {*base_set*(T) = empty or conflict exists}; $S(T)$:= *sq_no*++; enset(*h_list*, $S(T)$); copy(*h_list*, $H(T)$); check-local(T);

S_4: enqueue(dq_i, T); continue at S_3;

S_5: enqueue(q_1, T); continue at S_3;

S_6: prepa_mes($MESS_1$, T);

S_7: check-empty(q_1); dequeue(q_1, T); check-hole(T); check-local(T);

S_8: enset(U_b, T); continue at S_7;

S_9: read(T); compute(T); prepa_mes($MESS_2$, T); continue at S_{10};

S_{10}: write(T); check-empty(U_b);

S_{11}: loop: check-blocked(T, T_b); enqueue(q_1, T_b); until T_b = empty;

S_{12}: loop: d := min($base_set(T)$); deset($base_set(T)$, d); release-dataitem(d_i); enqueue(q_0, dequeue(dq_i)); until $base_set(T)$ = empty; deset(h_list, $S(T)$); commit(T); continue at S_7;

The events are as follows: e_0: local new update transaction has arrived; e_1: update request from other site has arrived; e_2: q_0 is not empty; e_3: conflict exists; e_4, e_6, e_8: continuing; e_5: T is local transaction; e_7: T is not local transaction; e_9: q_0 is empty; e_{10}: q_1 is not empty; e_{11}: T is blocked; e_{12}: continuing; e_{13}: T is local transaction; e_{14}: T is not local trahsaction; e_{15}: continuing; e_{16}: there are blocked transactions; e_{17}: there are no blocked transactions; e_{18}, e_{19}: continuing; e_{20}: there are no any transaction in the site; e_{21}: lock request from other site has arrived.

The resulting central site model for the centralized locking algorithm is shown in Figure 4.3.

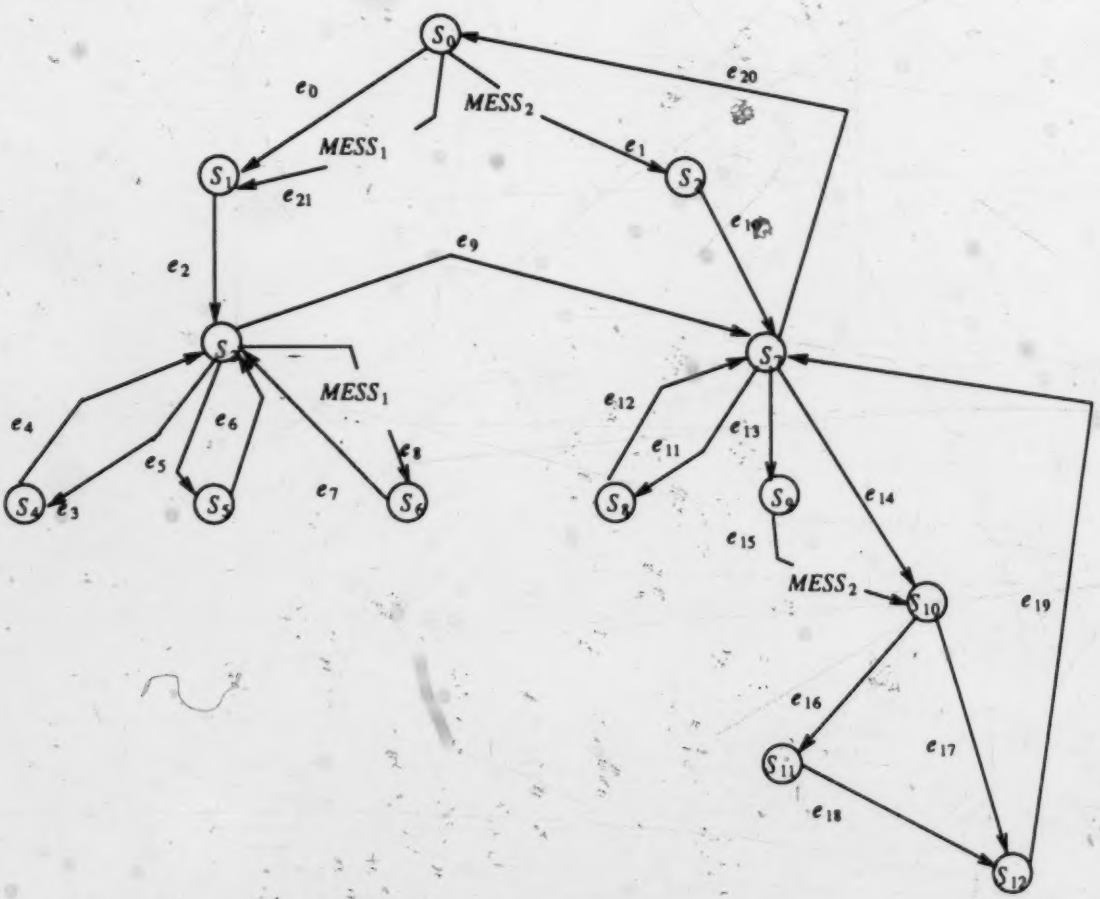


Figure 4.3 Central site model for the centralized locking algorithm.

4.2.1.2. Other Site Model

In the sites other than the central one, the model is as follow:

S_n: states = $\{S_0, S_1, S_2, \ldots, S_6\}$

E_n: events = $\{e_0, e_1, e_2, \ldots, e_{11}\}$

M_n: message types = $\{MESS_1, MESS_2\}$

The interpretation of message types is identical to the central site model.

The local data structures are presented as follows:

D_n: local data structures = {

d_i: data item i; i = 1, 2, ..., database size;

U_b: the set of blocked transactions;

q_0: queue for ready updating transactions;

T: transaction currently being executed;

base_set(T): base_set of transaction T;

$H(T)$: the hole list of transaction T;

$S(T)$: the sequence number of transaction T }

The local local operations are presented as follows:

L_n: local operations = {

enqueue(q, X): put X into queue q;

dequeue(q, X): remove X from queue q;

enset(U, X): put X into set U;

deset(U, X): remove X from set U;

read(T): read items from local database required by transaction T from *read_set*(T);

compute(T): compute new value for transaction T;

write(T): write new value for transaction T into local database;

check-hole(T): check-hole(T): check if all transactions with lower sequence number than T's sequence number, but that are not in the $H(T)$, have completed at this site;

check-empty(X): check if queue X or set X is empty;

check-local(T): check if transaction T is local;

check-blocked(T, T_b): check if there is a transaction in U_b blocked by T, if yes, T_b = blocked transaction, if not T_b = empty;

commit(T): T commits;

prepa_mes($Mess$, T): preparing message of $Mess$ type according to T }

Now we need to specify the events and states.

State: (in state S_i, the local operations that takes place are)

S_0: idle;

S_1: prepa_mes($MESS_1$, T);

S_2: enqueue(q_0, T); continue at S_3;

S_3: check-empty(q_0); check-local(T);

S_4: dequeue(q_0, T); read(T); compute(T); prepa_mes($MESS_2$, T); continue at S_5;

S_5: write(T); check-empty(U_b);

S_6: loop: check-blocked(T, T_b); enqueue(q_1, T_b); until T_b = empty; commit(T); continue at S_3;

The events are as follows: e_0: local new update transaction has arrived; e_1: lock request granted message has arrived; e_2: update request from other site has arrived; e_3: continuing; e_4: q_0 is not empty; e_5: T is local transaction; e_6: T is not local transaction; e_7, e_9: continuing; e_8: there are blocked transactions; e_{10}: there are no blocked transactions; e_{11}: there are no any transaction in the site.

The resulting other site model for the centralized locking algorithm is shown in Figure 4.4.

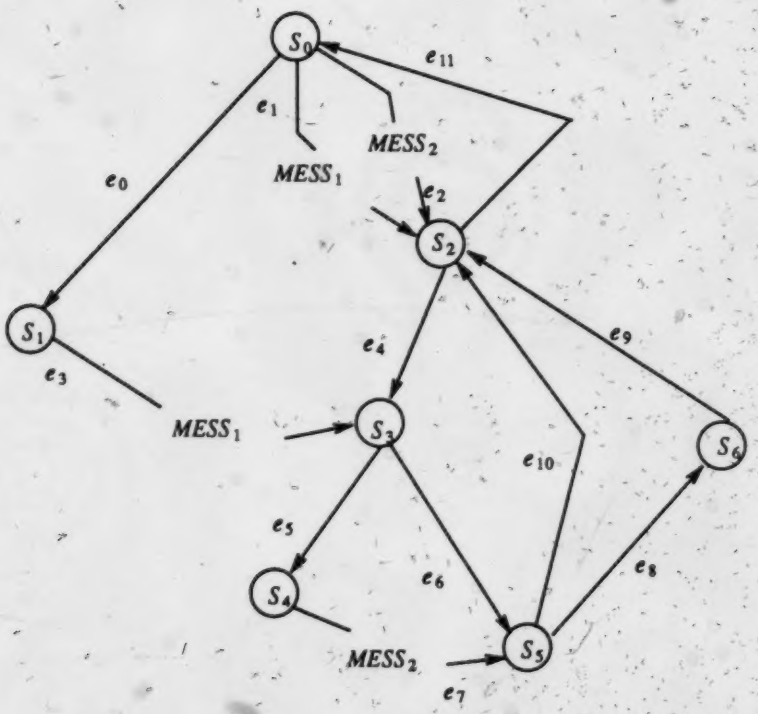


Figure 4.4 Other site model for the centralized locking algorithm.

4.2.2. Distributed Locking Algorithm

The distributed locking algorithm [GaC80] uses both locks and timestamps to synchronize the execution of transactions. The algorithm uses locking at each site to ensure the consistency of the local database, while using timestamps to ensure the mutual consistency of the multiple database copies.

The timestamps consist of time obtained from a local clock concatenated with the local site number. Each site keeps a local lock table for each transaction which contains the data items that need to be locked for each operation (e.g., read, update, etc.) of the transaction. The algorithm, as originally described , is designed to be resilient to system failures. In this analysis study, we will assume, for simplicity, that system failures do not

occur, and discuss a modified version of the algorithm.

When a transaction is generated, it is assigned a timestamp. The site where the transaction is originated will be referred to as the initiating site. The initiating site then broadcasts the transaction's timestamp and its base-set to all the sites. When this message is received, the sites execute a local locking procedure which consists of checking if the data items referenced in this transaction have been locked by other transactions and take one of the following two steps.

Step 1. If the data items have previously been locked, then there is a conflict and the site compares the timesamp of the conflicting transaction to that of the present transaction. If the conflicting transaction has a more recent timestamp, then the site aborts the conflicting transaction. Otherwise, it aborts the current transaction and it needs to be resubmitted later.

Step 2. If the data items have not been previously locked, then there are no conflicts, and there is no need to do anything.

In either of these cases, once the conflict is resolved, the site sends a message to the initiating site indicating that the lock request has been granted. When the initiating site receives all the grant messages from all the other sites, it broadcasts the update message, updates its copy of the database
and releases the locks for that transaction. Upon receipt of the update message, each site performs the update on its local copy of the database and releases the locks.

4.2.2.1. Model for The Distributed Locking Algorithm

The data management software to implement this algorithm is identical in every site, enabling the modeling of the algorithm as a single entity.

S_a: states = $\{S_0, S_1, S_2, \ldots, S_{10}\}$

E_a: events = $\{e_0, e_1, e_2, \ldots, e_{16}\}$

M_a: message types = $\{MESS_1, MESS_2\}$

Here, messages of type $MESS_1$ can be sent only to one destination site and messages of type $MESS_2$ can be sent to all the sites.

The local data structures are presented as follows:

D_a: local data structures = {

d_i: data item i; i = 1, 2, ..., database size;

$datimsp_i$: the timestamp of transaction which currently locks d_i;

T: transaction currently being executed;

$base_set(T)$: base_set of transaction T;

T_timsp: the timestamp of transaction T }

The local local operations are presented as follows:

L_a: local operations = {

assign($timsp, T$): assign a global unique timestamp to T;

read(T): read items from local database required by transaction T from $read_set(T)$;

compute(T): compute new value for transaction T;

write(T): write new value for transaction T into local database;

check-all(T): check if the lock granted messages have arrived from all the sites;

check-local(T): check if transaction T is local;

lock-dataitem(d_i): lock data item d_i;

older($timsp_1$, $timsp_2$): check two timestamps;

release-dataitem(T): release all data item locked by T;

prepa_mes($Mess$, T): preparing message of $Mess$ type according to T;

abort(T): abort T;

commit(T): T commits}

Now we need to specify the states and events in terms of the local operations.

State: (in state S_i, the local operations that takes place are)

S_0: idle;

S_1: assign($timsp$, T); continue at S_6;

S_2: continue at S_6;

S_3: check-all(T);

S_4: commit(T); release-dataitem(T);

S_5: prepa_mes($Mess_2$, T);

S_6: loop(base_set (T): {older(T_timsp, $datimsp_i$); lock-dataitem(d_i);}; check-local(T);

S_7: read(T); compute(T); prepa_mes($MESS_2$, T); continue at S_8;

S_8: write(T); continue at S_4;

S_9: prepa_mes($MESS_1$, T);

S_{10}: abort(T); release-dataitem(T);

The events are as follows: e_0: local new update transaction has arrived; e_1: lock request from other site has arrived; e_2: lock granted message has arrived; e_3: update request from

other site has arrived; e_4, e_5: continuing; e_6: T is not local transaction; e_7: T is local transaction; e_8: conflict exists; e_9: current transaction is aborted; e_{10}: lock request is granted; e_{11}: all lock requests from all other site have arrived; e_{12}: continuing; e_{13}: all lock requests from all other site have not arrived; e_{14}: continuing; e_{15}: current transaction commits e_{16}: all lock requests are granted in its own sites.

The resulting site model for the distributed locking algorithm is shown in Figure 4.5.

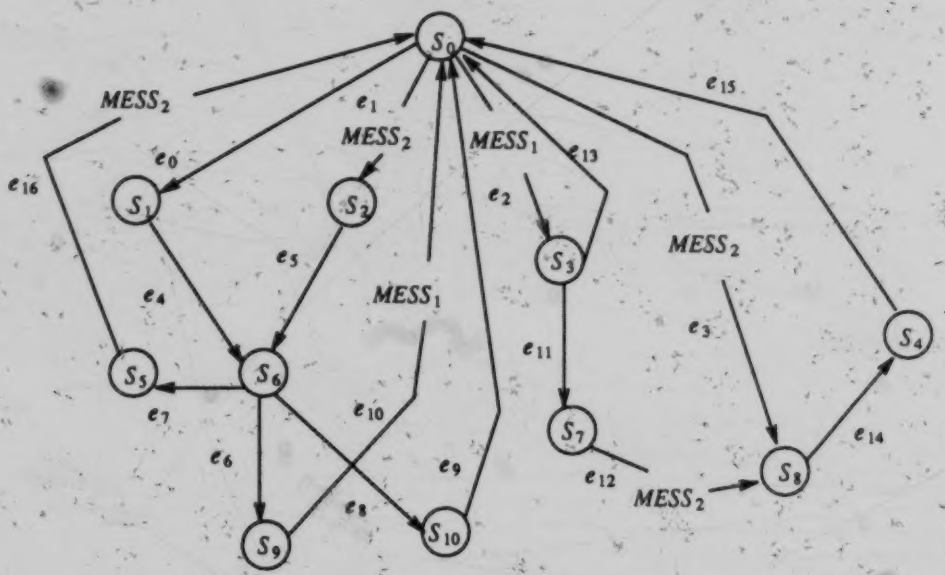


Figure 4.5 Model for the distributed locking algorithm.

4.3. The Local Area Network Model

As mentioned before, we have selected Ethernet and Token Ring as the networks over which the concurrency control algorithms are implemented. In this section, we discuss LANSF models of the protocols employed by these network.

4.3.1. The Ethernet Protocol

Under Ethernet protocol the network consists of a number of sites (called "stations" in LANSF) distributed along the bus. The bus is a single link and forms a tree with the stations being the leaves. Each station has a single port connecting it to the bus. All stations constantly monitor the bus and are able to perceive its status as busy or idle. A station acquiring a packet to transmit senses the bus. If the bus has been idle for the amount of time equal to the inter-packet space, the station starts transmitting the packet. Otherwise, the station waits until it becomes idle, obeys the inter-packet spacing constraint and transmits. A transmitting station keeps sensing for interference caused by some other activity on the bus. If collision is detected, the station immediately aborts the transfer and sends a jamming signal to make sure that any other party involved in the collision also recognizes it. Then the station reschedules the retransmission.

The algorithm used to reschedule the retransmission after a colliding transfer attempt is the so-called Binary Exponential Backoff. The amount of waiting time is determined as:

$$2t_a * U\,(0.2^{cc-1}),$$

where t_a is the maximum end-to-end propagation delay of the bus (256 bit), cc is the collision counter and U stands for a uniformly distributed random number from the given interval. The standard function I_uniform generates a uniformly distributed random number between the values of the arguments, inclusively. We assume that the collision counter never grows above 10.

Ethernet is a uniform network, in the sense that all stations obey the same protocol and transmit with the same speed. Thus, it is natural and convenient to assume that all distances and time intervals are expressed in bits. The numerical values of the relevant parameters for the network described above are given below. Note that all values are

expressed in bits.

Data rate	10 Mbps
Minimum packet size	368 bits
Maximum packet size	12000 bits
Frame information	208 bits
Minimum packet spacing	96 bits
Minimum jam size	32 bits
Maximum bus length	256 bits

The Ethernet protocol can be modeled in two parts by using the following commands embedded in LANSF [Gbu90] as below:

```
receiver () {
 int *count;
 switch (the_action) {
  case INITIALIZE:
  case WAIT_FOR_PACKET:
   wait_event (BUS, END_MY_PACKET, PACKET_RECEIVED);
   return;
  case PACKET_RECEIVED:
   count = memreq (sizeof(int));
   *count = 1;
   queue_item (the_station->id, count, RECEIVE_QUEUE);
   skip_and_continue_at (WAIT_FOR_PACKET);
 }
}

transmitter () {
TIME   last_silence, idle_period, backoff ();
 switch (the_action) {
  case INITIALIZE:
   *BUFFER = make_packet (NULL, NULL, NULL);
  case NEXT_PACKET:
   if (!is_item (TRANSMIT_QUE_1)) {
    wait_event (QUEUE, TRANSMIT_QUE_1, NEXT_PACKET);
    return; }
case PACKET_LENGTH:
 the_station->Tran = get_item (TRANSMIT_QUE_1);
 if (the_station->Tran->status_T == Lock_Req_T)
  BUFFER->length = SHORT_PACKET;
 else
  BUFFER->length = LONG_PACKET;
 BUFFER->receiver = id_to_station (the_station->Tran->rcv_site_id);
 the_station->curr_time = current_time;
```

```
the_station->collision_counter = 0;
        case RETRY:
          last_silence = last_eoa_sensed (BUS);
          if (undef (last_silence)) {
          wait_event (BUS, SILENCE, RETRY);
          return;
          }
          idle_period = minus (current_time, last_silence);
          if (les (idle_period, space)) {
          wait_event (TIMER, minus (space, idle_period), TRANSMIT);
          return;
          }
        case TRANSMIT:
          transmit_packet (BUS, BUFFER, END_PACKET);
          wait_event (BUS, COLLISION, ABORT_PACKET);
          return;
        case END_PACKET:
          stop_transfer (BUS);
          queue_item (the_station->Tran->rcv_site_id, the_station->Tran,
              PENDING_QUEUE);
          continue_at (NEXT_PACKET);
        case ABORT_PACKET:
          abort_transfer (BUS);
          the_station->collision_counter ++;
          emit_jam (BUS, jam, STOP_JAM);
          return;
        case STOP_JAM:
          end_jam (BUS);
          wait_event (TIMER, backoff (), RETRY);
          return;
      }
    }


    TIME  backoff () {
      long        mback, cc;
    #define LOG2    0.69314718055994530942  /* Natural logarithm of 2 */
      if ((cc = the_station->collision_counter) > 10) cc = 10;
      mback = exp ((double)cc * LOG2) - 0.5;
      return (multi (round_trip, l_uniform (0, mback)));
    }
```

4.3.2. The Token Ring

In a ring-type network, stations are connected into a circular structure in such a way that each station is directly attached to its two immediate neighbors via two separate ports. With the token ring protocol, information is passed along the ring in one direction, e.g. clockwise. Thus, each station receives information on one (called "input") port and transmits on the other (called "output") port. A special packet (token) is circulated among the stations. If the station receiving the free token has no data packet awaiting transmission, it immediately relays the token by copying it to the output port. Otherwise, the station captures the free token, changes it to a busy token and passes it on, followed by the data for a destinaion station. After the busy token returns to its origin, having traversed the ring, the originating station releases a new free token for another station to capture and to carry on the process. A station that transforms a free token to a busy token waits until the busy token returns before releasing a free token. A station may only transmit one data packet per each acquisition of the token. A natural way of modeling the protocol in LANSF is to have two parts at each station, one part servicing the input port and processing packets arriving to the station, the other part handling the output port and passing packets to the station's successor.

The token ring can be modeled in following two parts by using the following commands embedded in LANSF [Gbu90] as below:

```
input_process () {
 int i, *count;
 switch (the_action) {
  case INITIALIZE:
   the_station->acting = 0;
   if (the_station->id == 0) {
    make_token ();
```

```
    case GOT_TOKEN:
      internal_signal (TRANSFER_PERMITTED);
    case WAIT_FOR_SILENCE:
      wait_event (INPUT_PORT, SILENCE, WAIT_FOR_PACKET);
      return;
     }
    case WAIT_FOR_PACKET:
     wait_event (INPUT_PORT, ACTIVITY, RECEIVING_PACKET);
     return;
    case RECEIVING_PACKET:
     *RELAY_BUFFER = *the_packet;
     wait_event (DELAY, hdr, HEADER_RECOGNIZED);
     return;
    case HEADER_RECOGNIZED:
     if (RELAY_BUFFER -> type == FREE_TOKEN)
      continue_at (GOT_TOKEN)
     else
      if (RELAY_BUFFER -> type == BUSY_TOKEN) {
       if (the_station->acting == 1) {
        the_station->acting = 0;
        RELAY_BUFFER -> type = FREE_TOKEN; }
       internal_signal (PASSING_TOKEN);
       continue_at (WAIT_FOR_SILENCE); }
     if (my_packet (RELAY_BUFFER)) {
      i = RELAY_BUFFER->length - hdr;
      wait_event (DELAY, i,  PACKET_RECEIVED);
      return;
     }
     internal_signal (PACKET_TO_PASS);
     continue_at (WAIT_FOR_SILENCE);
    case PACKET_RECEIVED:
     count = memreq (sizeof(int));
     *count = 1;
     queue_item (the_station->id, count, RECEIVE_QUEUE);
     continue_at (WAIT_FOR_SILENCE);
    default:  excptn ("Left_port_process: illegal action");
   }
}

output_process () {
 switch (the_action) {
  case INITIALIZE:
    *PACKET_BUFFER = make_packet (NULL, NULL, PACK_LENG);
  case WAIT_SIGNAL:
   wait_event (SIGNAL, TRANSFER_PERMITTED,
         TRANSMIT_OWN_PACKET);
   wait_event (SIGNAL, PACKET_TO_PASS,
         TRANSMIT_OTHER_PACKET);
   wait_event (SIGNAL, PASSING_TOKEN,
         PASS_TOKEN);
```

```
    return;
  case TRANSMIT_OWN_PACKET:
    if (is_item (TRANSMIT_QUE_1)) {
      the_station->Tran = get_item (TRANSMIT_QUE_1);
      PACKET_BUFFER->receiver
                = id_to_station (the_station->Tran->rcv_site_id);
      the_station->acting = 1;
      RELAY_BUFFER -> type = BUSY_TOKEN;
      transmit_packet (OUTPUT_PORT, RELAY_BUFFER, WAIT_TO_TRANS);
      return;
    } else {
  case PASS_TOKEN:
     transmit_packet (OUTPUT_PORT, RELAY_BUFFER, TOKEN_PASSED);
     return;
    }
  case WAIT_TO_TRANS:
    stop_transfer (OUTPUT_PORT);
    wait_event (DELAY, spc, TRANSMITTING);
    return;
  case TRANSMITTING:
    transmit_packet (OUTPUT_PORT, PACKET_BUFFER,
              PACKET_TRANSMITTED);
    return;
  case TOKEN_PASSED:
  case TRANSFER_COMPLETED:
    stop_transfer (OUTPUT_PORT);
    continue_at (WAIT_SIGNAL);
  case PACKET_TRANSMITTED:
    stop_transfer (OUTPUT_PORT);
    queue_item (the_station->Tran->rcv_site_id, the_station->Tran,
              PENDING_QUEUE);
    continue_at (INITIALIZE);
  case TRANSMIT_OTHER_PACKET:
    transmit_packet (OUTPUT_PORT, RELAY_BUFFER,
              TRANSFER_COMPLETED);
    return;
  default:  excptn ("Right_port_process: illegal action");
  }
}

make_token () {
  *RELAY_BUFFER = make_packet (NULL, NULL, tkl);
  RELAY_BUFFER -> type = TOKEN_PACKET;
}
```

The numerical values of the relevant parameters for the network described above are given below.

Data rate	4 Mbps
Minimum packet length	368 bits
Maximum packet length	12000 bits
Header	176 bits
Trailer	32 bits
Token length	176+32 bits (frame only)
Packet space	96 bits
Token passing timeout	20000 bits

4.4. Input Parameters

The model input parameters are:

1. *Mean interarrival time of transactions (I_t):* The arrival time of transaction to each site is considered to be negative exponentially distributed with a mean interarrival time I_t.

2. *Mean base-set size (B_t):* The base-set represents the set of items that are referenced by a transaction. The base-set size is assumed to be negative exponentially distributed with a mean base-set size of B_t. The write-set size refers to the set of items that are modified by a transaction. The write-set size is assumed to be uniformly distributed between 1 and the base-set size.

3. *Number of sites in the system (N):* The number of sites in the system is assumed to be N.

4. *CPU time slice (C_s):* The CPU time slice is the time it takes a CPU server to do some computations, for example, to modify or check a lock or to compare two values.

5. *CPU compute time (C_c):* This is the amount of time that is needed per base set item to compute a new update value. For example, if B is the base set size of a transaction, the total time required to compute all the update values for the transac-

tion is $B * C_e$.

6. *I/O time slice (IO_s):* The I/O time slice is the time needed to read or write a lock or a timestamp from/to the secondary storage.

7. *I/O update time (IO_e):* This is the time it takes to read or write an item from of to the secondary storage.

8. *Database size (D):* The database size is the total number of data items in the database.

9. *Date rate (D_r):* Network transmission rate.

The ranges of values that these parameters assume are as follows: I_t: 0.5, 1, 5, 10, 15 s, B_t: 2, 5, 10 data items, N: 5, 10, 50, 100 sites, D: 10, 100, 500, 1000, 1200, 1500 data items, D_r: 1, 4, 10, 16, 50, 100 Mbps.

The default (typical) values for all the parameters, on the other hand, are as follows: I_t = 5s, B_t = 5 data pages, N = 50 sites, C_e = 0.001 s, C_s = 0.00001 s, IO_e = 0.025 s, IO_s = 0 s, D = 1000 data items, D_r = 10 Mbps for Ethernet, and D_r = 4 Mbps for Token Ring.

4.5. Performance Metrics

The performance metrics that are observed during each simulation run include the following:

1. *Mean response time:* The response time of a transaction is defined as the interval of time between the arrival of the transaction at a site and its completion at the same site. A transaction is assumed to be completed when the originating site has successfully done all the work requested by the transaction. The mean response time is the average of the response times for all successfully completed transactions.

2. *I/O, CPU and Network utilizations:* The utilization of a server at a site is defined as the percentage of the available time that the server is busy.

3. *Throughput:* In this study, the throughput in system is the ratio of the total number of transactions committed in the system to the simulation time.

4 *Mean conflict:* In this study, the mean conflict is the ratio of the total number of transactions entering the system to the total number of conflicts between transactions. So, in CL, it means the average no. of conflicts per transaction, but in DL it may reflect the percentage of transactions which are aborted.

4.6. Design Considerations

LANSF provides a very powerful tool to investigate the behavior of networks at the media access level. It is necessary to produce additional source code to implement the higher level functionality called for by the concurrency control algorithms in order to have have a realistic simulation.

Consideration should be given to the design at all levels to try to directly simulate the low level processes involved. In certain cases, it is necessary to characterize the behavior of some processes to produce the simulator. For example, because the major concern of this thesis is to examine how the performance of distributed database transaction management is affected by the underlying local area network architecture, there is no attempt to fully simulate the data processing at the physical level. It should be noted that this does not affect the timing information generated by the simulation. Simulated delays should still represent the behavior of the system.

As a further step towards making these results compatible with the previously obtained ones, certain restrictions are imposed on the models. These restrictions and further analysis related assumptions can be listed as follows:

1. The base-set of the transactions are known when they are originated so that the size of the base-set can be statistically determined.
2. Most of the transactions have a small base-set with a few transactions with large base-sets. Thus, the base set size can be approximated by an exponential distribution.
3. No failures occur in the system.
4. There is no overlapping of CPU and I/O activities.
5. The database is static. Thus, once the database size is fixed, there are no additions or deletions to the database.
6. The database is fully duplicated, thus eliminating the need to consider the effects of transaction processing and directory management.

The database sizes used (100-1500 lockable items) may not be realistic. However, the lockable items may be physical pages, in which case the database becomes of the reasonable size. Furthermore, the effect of the database size diminishes as it gets larger. This is because the probability of conflict gets smaller as the database gets larger.

4.7. Simulating Environment

The simulation system presented in this thesis is developed on the LANSF (Local Area Network Simulation Facility) simulator package [Gbu90]. The experiments are run on a MIPS running RISC/OS.

Chapter 5
Simulation Results

The simulation results are presented and analyzed in this chapter. In the remainder, the centralized locking algorithm will be referred to as "CL" and the distributed locking algorithm will be referred to as "DL". First we discuss the performance of CL over Ethernet and token ring and then we will compare the results of DL over Ethernet and token ring.

5.1. CL over Ethernet and Token Ring

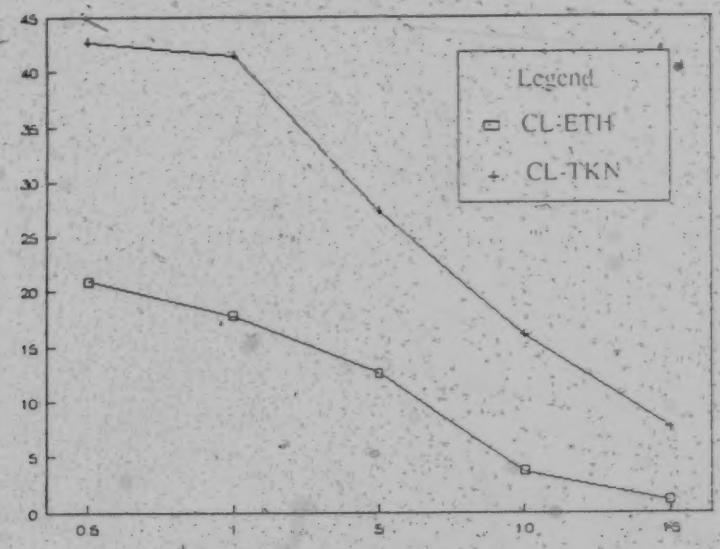
5.1.1. Effect of Interarrival Time

Figure 5.1 shows the effect of interarrival time on network utilization. We notice that the utilization of both Ethernet and token ring increases as system loads get heavy and also token ring gives higher utilization. In our study, network utilization can also be interpreted as network loads. The difference is caused by the way that message passing is performed. Recall that in token ring, the information is passed along the ring in one direction, e.g. clockwise. In CL, every transaction sends lock request message to central site and central site sends lock granted message back. The whole message has to go around the ring. But for the bus type network, such as Ethernet, the information can be transmitted in either direction.

Figure 5.2 depicts the response time as it varies with interarrival time. Under heavy load, CL over Ethernet suffers from what is known as the "central site bottleneck." As can be observed from Figures 5.4 and 5.5, under heavy load, I/O utilization and CPU utilization of CL over token ring are higher than those over Ethernet (note the utilization

figures also represent the overhead of the algorithm). Furthermore, the mean conflict over both network shows no big difference(Figure 5.6). Thus, the sharp upturn of response time of CL over Ethernet is explained by the overloading of the central site by lock request and subsequent message delay. In Ethernet, due to the probabilistic nature of the protocol, it may happen that two or more sites will collide for a period of time. The maximum observed message delay may grow indefinitely (theoretically it is unbounded) as loads increase in Ethernet, whereas for the token ring, it is always limited.

In Figure 5.3, the effect of interarrival time on throughput of system is shown. As can be observed, the effect is significant, with both of the network protocols exhibiting similar behavior. The lower values of CL algorithm over Ethernet under heavy load are primarily due to the same effect of longer message delay as mentioned above.



Interarrival Time (sec)

Figure 5.1 Effect of interarrival time on network utilization (%).

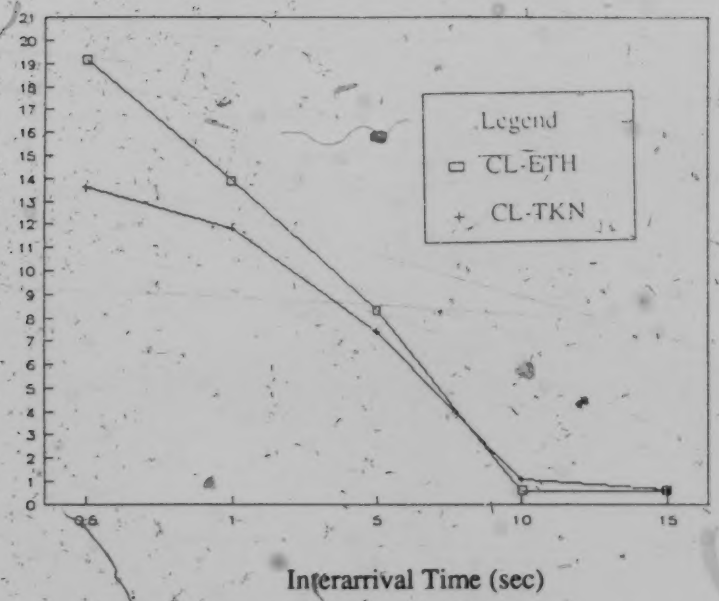


Figure 5.2 Effect of interarrival time on mean response time (sec).

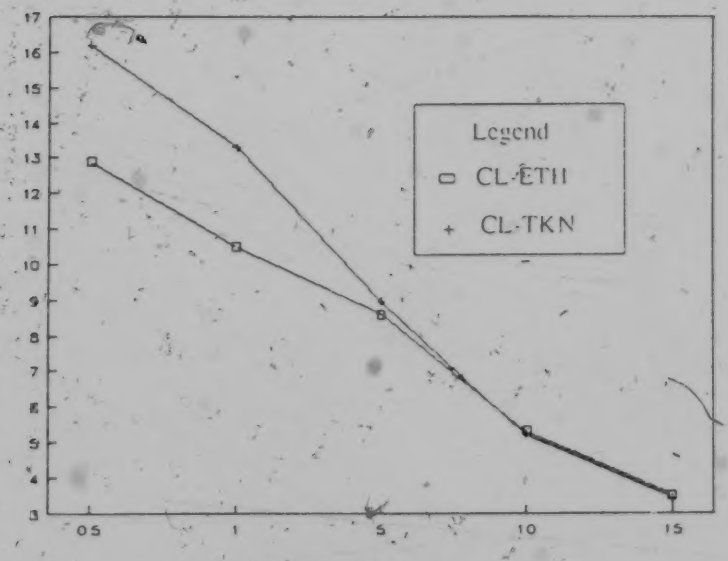


Figure 5.3 Effect of interarrival time on mean throughput.

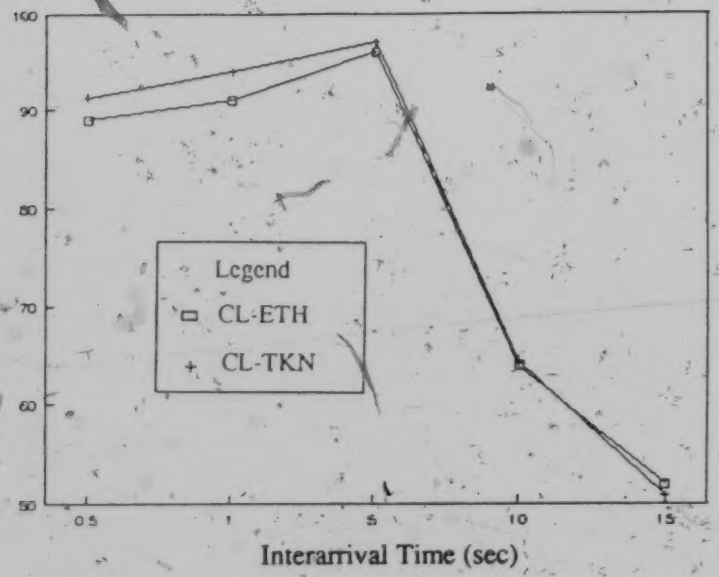


Figure 5.4 Effect of interarrival time on I/O utilization (%).

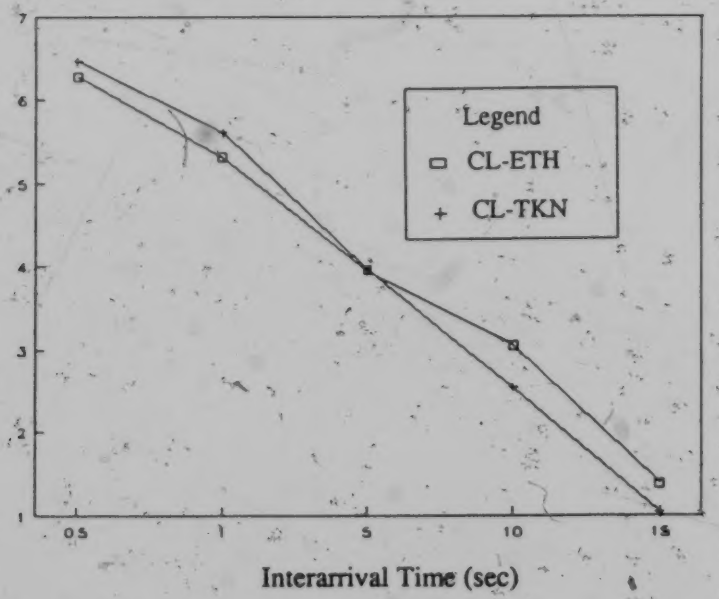


Figure 5.5 Effect of interarrival time on CPU utilization (%).

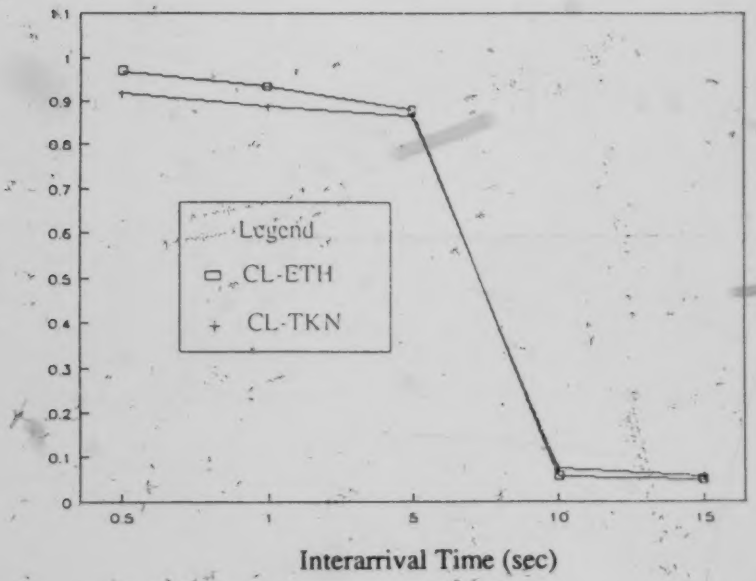


Figure 5.6 Effect of interarrival time on mean conflict (per transaction).

5.1.2. Effect of Date Rate

In Figure 5.7, the effect of network data rate on network utilization is shown. As can be observed, the network loads increase as network data rate decreases. This is mainly due to longer time needed for transmitting messages in the network. The sharp upturn of response time of CL over Ethernet under low data rate (high network loads) again can be observed in Figure 5.8. Before we draw any conclusions, we need to look at I/O utilization, CPU utilization and mean conflict (See Figures 5.10, 5.11 and 5.12). Notice that mean conflict is so low that the effect on the mean response time is negligible. Under heavy network loads, both I/O and CPU overheads under token ring is higher than that under Ethernet. Therefore, the sharp upturn of response time in Ethernet is again mainly due to the longer message delay in the network. The high variance of response time in Ethernet decreases throughput. This can be observed in Figure 5. 9.

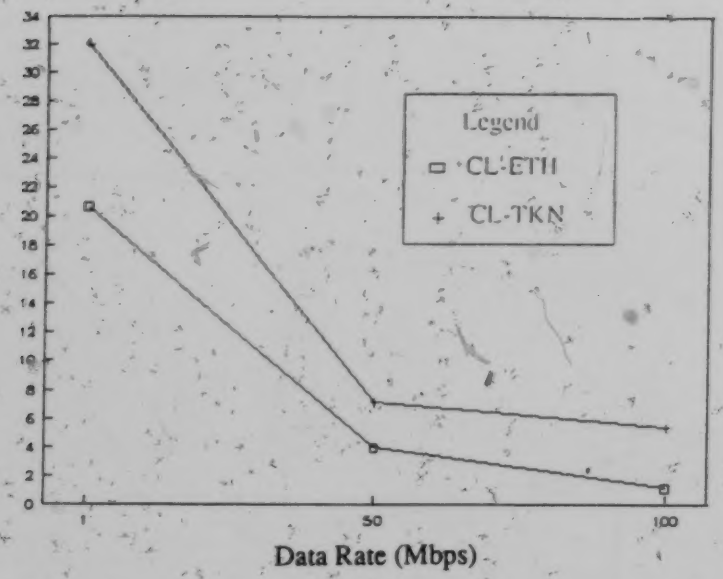


Figure 5.7 Effect of data rate on network utilization (%).

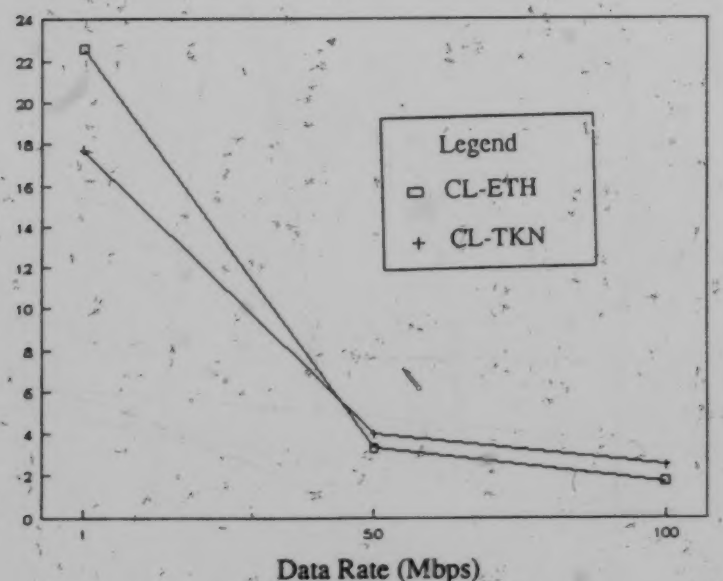


Figure 5.8 Effect of data rate on mean response time (sec).

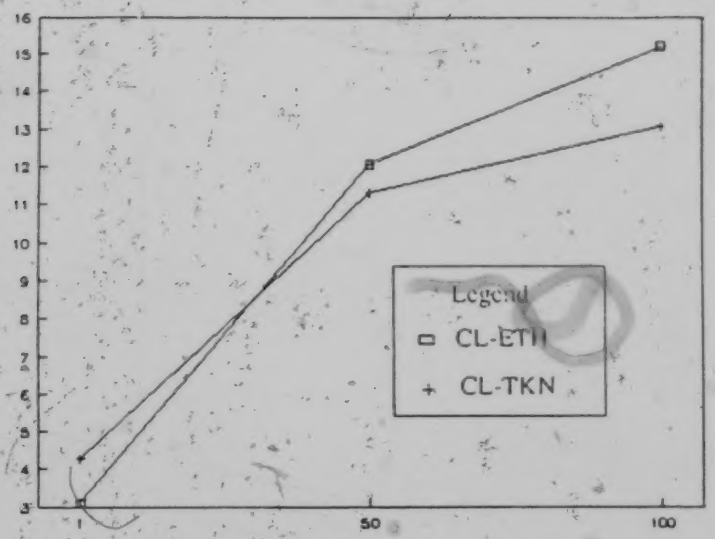


Figure 5.9 Effect of data rate on mean throughput.

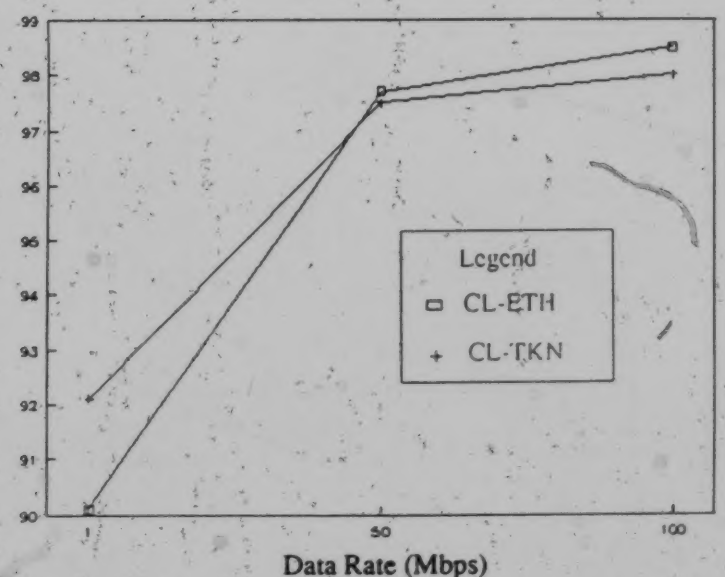


Figure 5.10 Effect of data rate on I/O utilization (%).

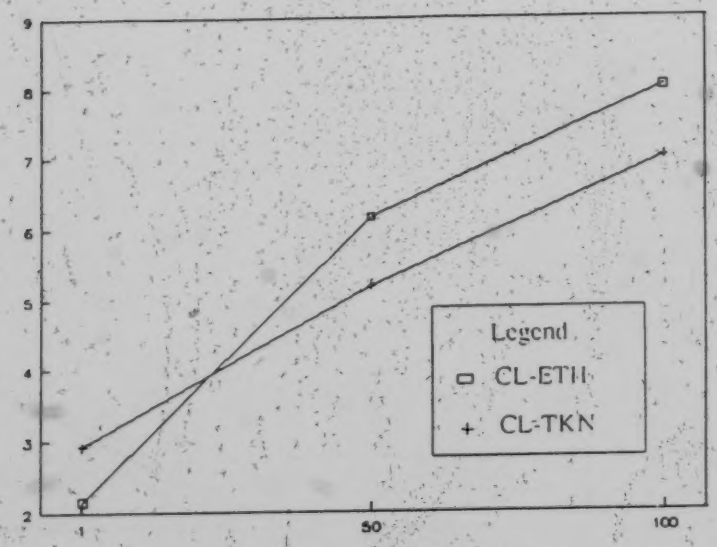


Figure 5.11 Effect of data rate on CPU utilization (%).

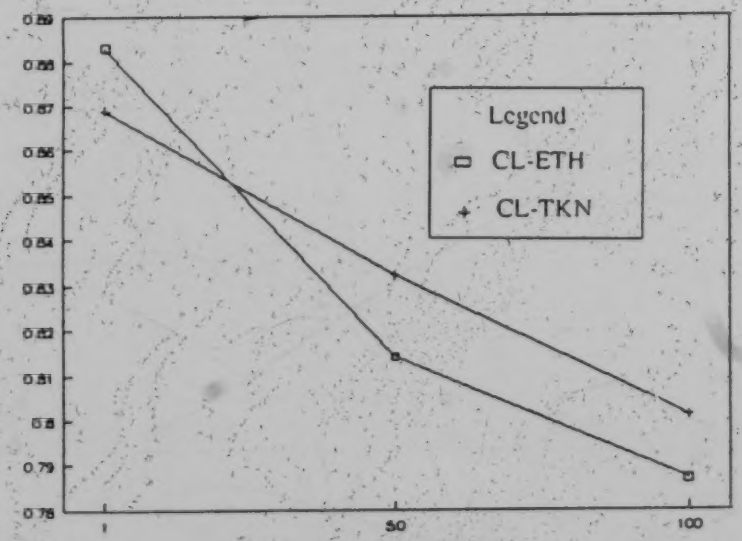


Figure 5.12 Effect of data rate on mean conflict (per transaction).

5.1.3. Effect of Number of Sites

The effect of number of sites on performance of CL over Ethernet and token ring in terms of mean transaction response time is shown in Figure 5.13. For both network protocols, increasing the number of sites in the system degrades the performance. The mean response time for both network protocols increases slowly at first until N reaches about 12 sites where it starts to rise rapidly. In both cases, increase in response time is due to the fact that as the number of sites increases the overall rate of arrival of transaction into the system also increases causing network contention and additional delays in the processing of the transactions. The sharp increase in Ethernet curve occurs as a result of the overloading of the central site by lock requests and the subsequent longer message delay due to the increasing probability of collisions between packets.

The throughput is also sensitive to the number of sites, which is depicted in Figure 5.14.

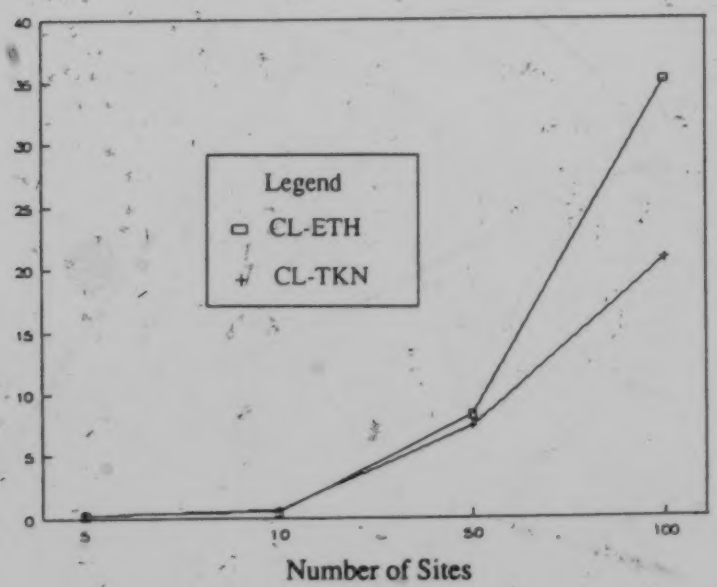


Figure 5.13 Effect of number of sites on mean response time (sec).

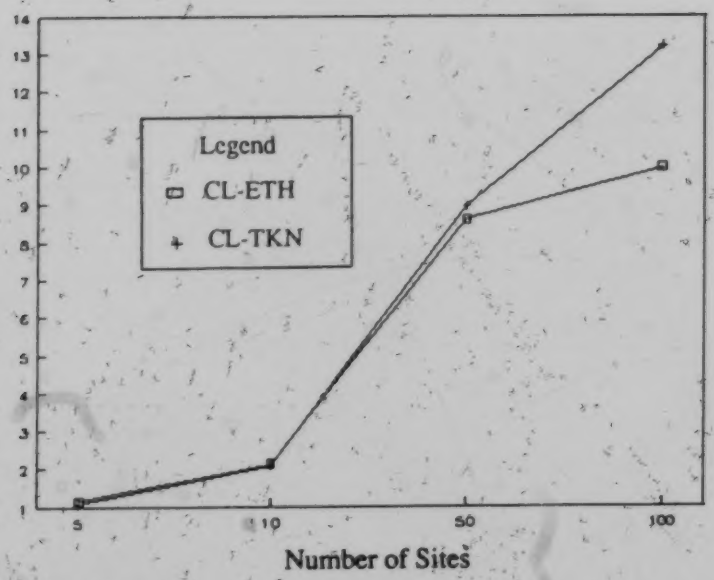


Figure 5.14 Effect of number of sites on mean throughput.

5.1.4. Effect of Base Set Size

For both network protocols, the effect of base set size on the mean response time and throughput is significant. These are shown in Figures 5.15 and 5.16. Recall that there are 50 sites in the system; thus increasing the base set size of transactions will cause additional delays in their processing. The rapid increase in mean response time is accounted for by the fact that the delays due to conflict are negligible when the base set size is small, but becomes more dominant as base set size gets large (See Figure 5.17).

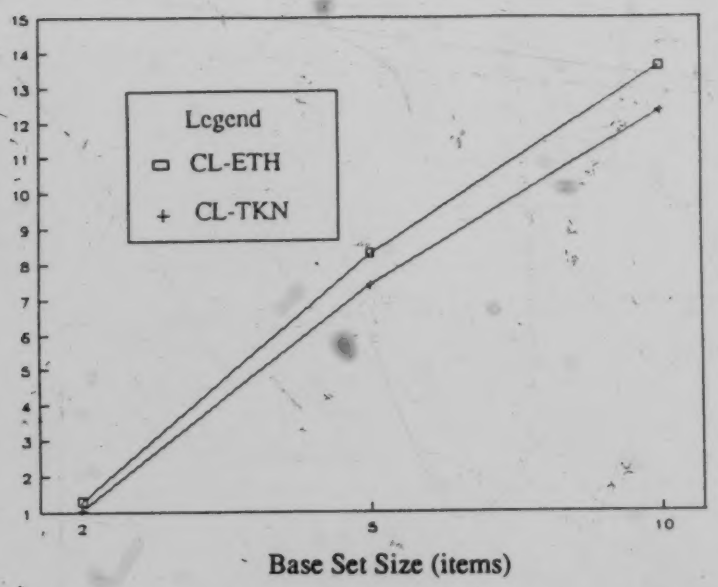


Figure 5.15 Effect of mean base set on mean response time (sec).

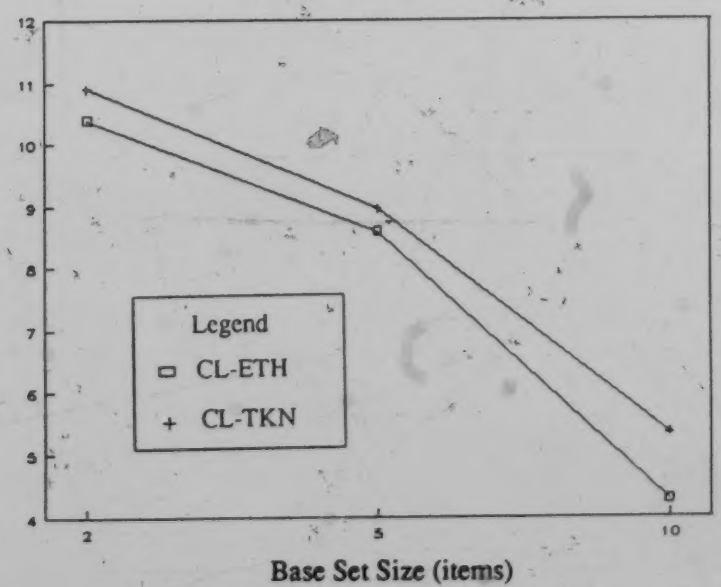


Figure 5.16 Effect of mean base set on mean throughput.

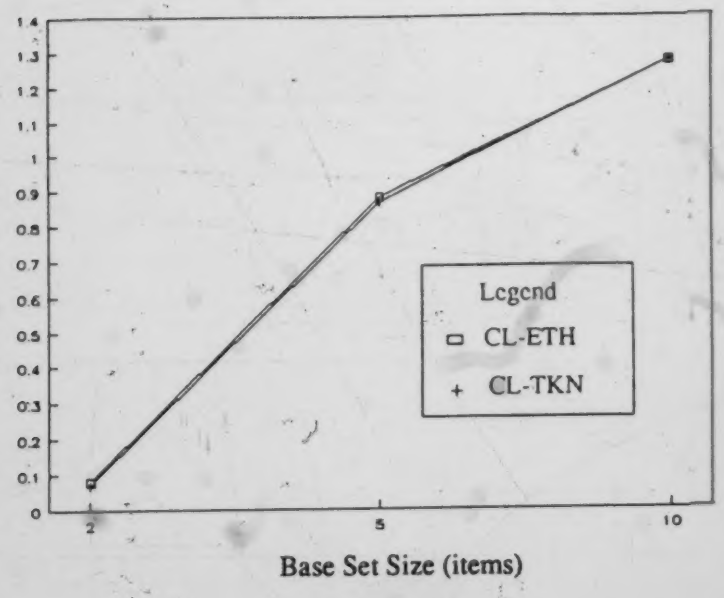


Figure 5.17 Effect of mean base set on conflict (per transaction).

5.1.5. Effect of Database Size

Figure 5.18 shows the effect of database size on the mean response time. For larger database sizes the response time for both Ethernet and token ring protocols is constant. As the number of items in the database is reduced below 500, the mean response time increases very rapidly. This is caused by the conflicts that occur when the transactions have to compete for a smaller number of items (See Figure 5. 19).

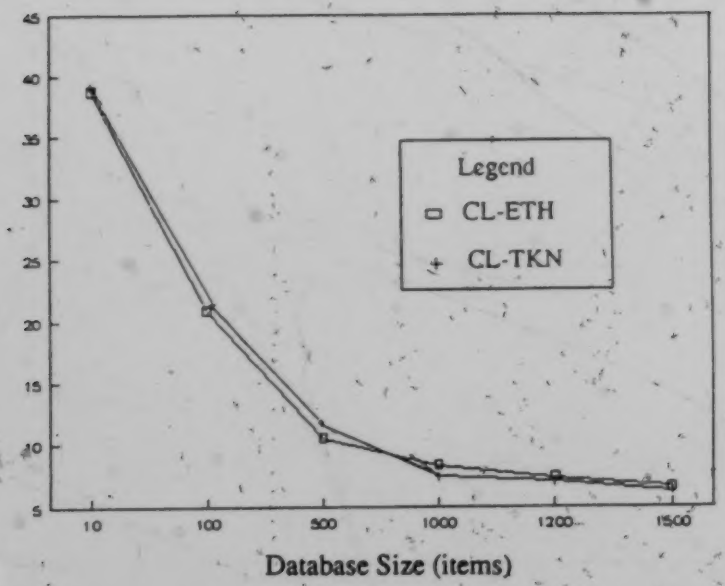


Figure 5.18 Effect of database size on response time (sec).

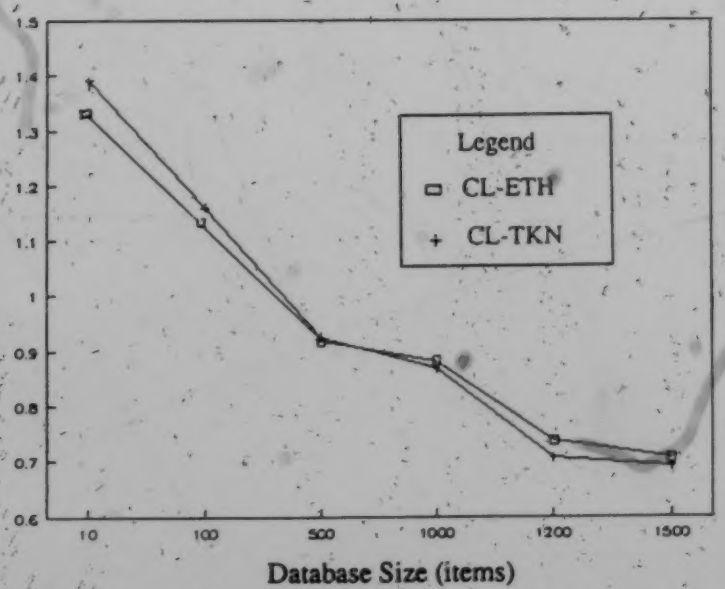


Figure 5.19 Effect of database size on conflict (per transaction).

5.2. DL over Ethernet and Token Ring

5.2.1. Effect of Interarrival Time

Figure 5.20 shows the effect of interarrival time on network utilization. We notice that the utilization for both Ethernet and token ring is large. This is because DL algorithm broadcasts lock requests and update requests to all the sites. As can be observed in Figure 5.21, under heavy system loads (which cause heavy network loads as mentioned above), the response time in token ring increases rapidly as expected. But the response time in Ethernet does not change too much. We have mentioned that in an Ethernet a packet that has experienced several collisions could suffer a longer delay than a packet that has experienced less or no collisions. Therefore, it is possible for an older transaction to experience longer delays than a younger one, especially if Ethernet is heavily loaded. This could cause a significant degradation in performance of timestamp based concurrency control algorithms. This is because many transactions having smaller timestamp values (i.e. older transactions) may attempt to read (write) data items having larger (i.e. more recent) read (write) time stamp values. A large number of transaction are aborted (See Figure 5.23). Thus the response times of the remaining transactions are reduced, but at the prices of reduced system throughput (See Figure 5.22). This shows that timestamp based algorithms are not suitable for heavily loaded contention type networks.

Token rings are better candidates for implementing timestamp based algorithms, because of their bounded delay. Under normal circumstances, even under heavy loads, too many transactions are not aborted in token rings.

In the following two sections, we have observed the same phenomena as above.

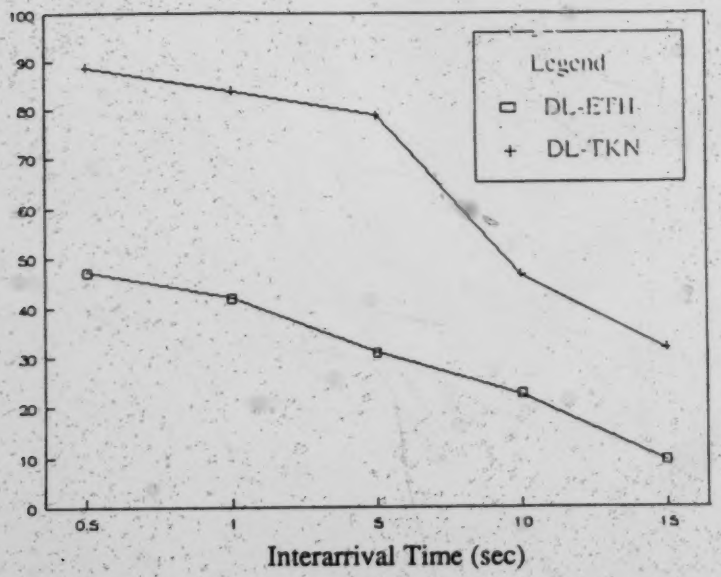


Figure 5.20 Effect of interarrival time on network utilization (%).

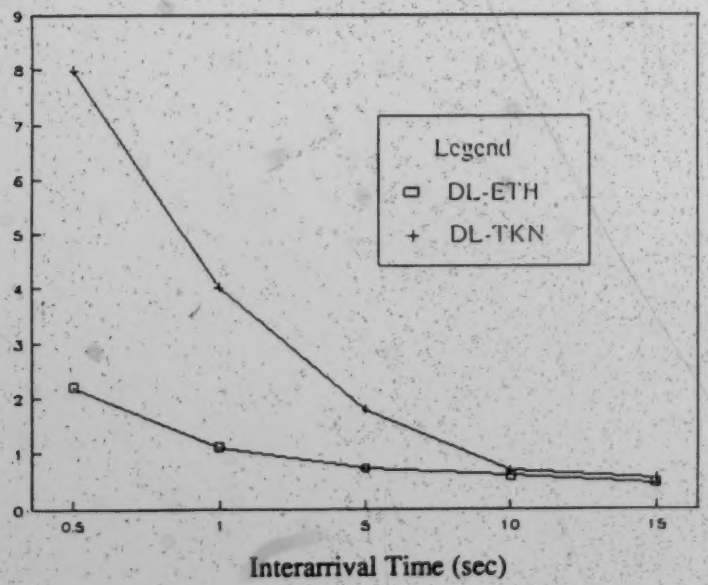


Figure 5.21 Effect of interarrival time on response time (sec).

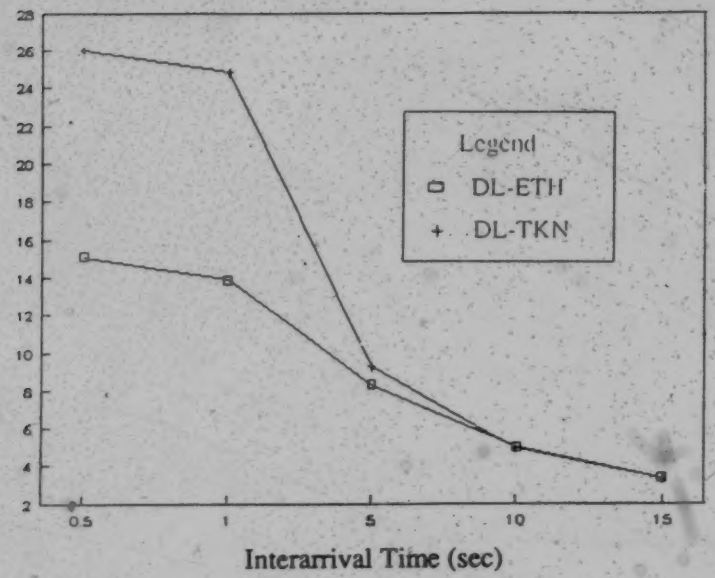


Figure 5.22 Effect of interarrival time on mean throughput.

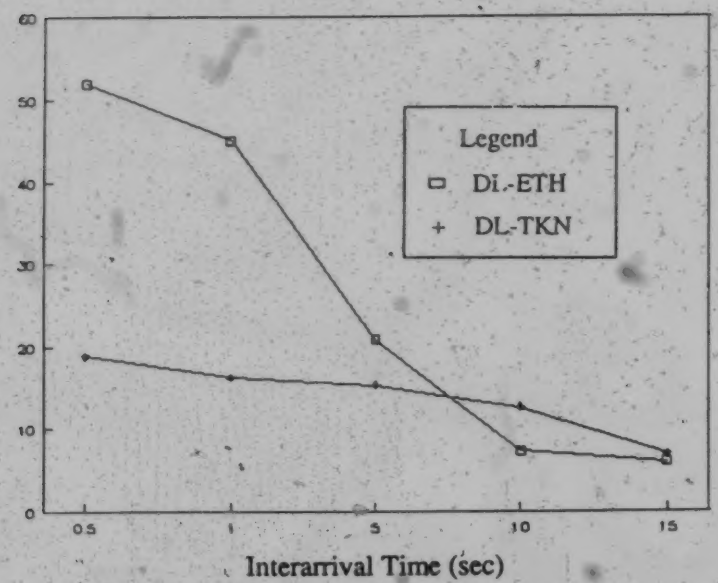


Figure 5.23 Effect of interarrival time on conflict (%).

5.2.2. Effect of Date Rate

In Figure 5.24, the effect of network data rate on network utilization is shown. As can be observed, the network loads increase as network data rate decreases. This is mainly due to longer time needed for transmitting messages in the network. The sharp upturn of response time of DL over token ring under low data rate (high network loads) again can be observed in Figure 5.25. As it takes longer to transmit messages between the sites under low data rate, the probability of collisions between packets in Ethernet will increase. This could cause a significant degradation in performance due to the same effect as above. Higher abortion rate and lower values of system throughput of DL over Ethernet also can be observed in Figures 5.26 and 5.27.

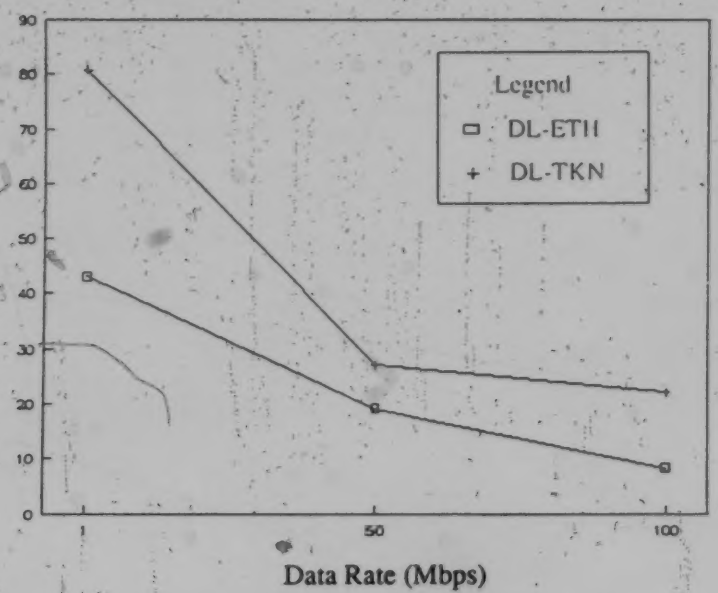


Figure 5.24 Effect of data rate on network utilization (%).

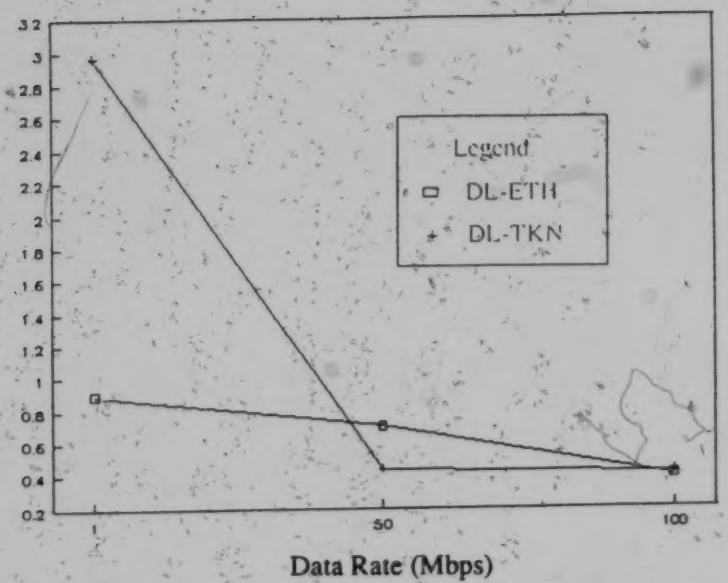


Figure 5.25 Effect of data rate on response time (sec).

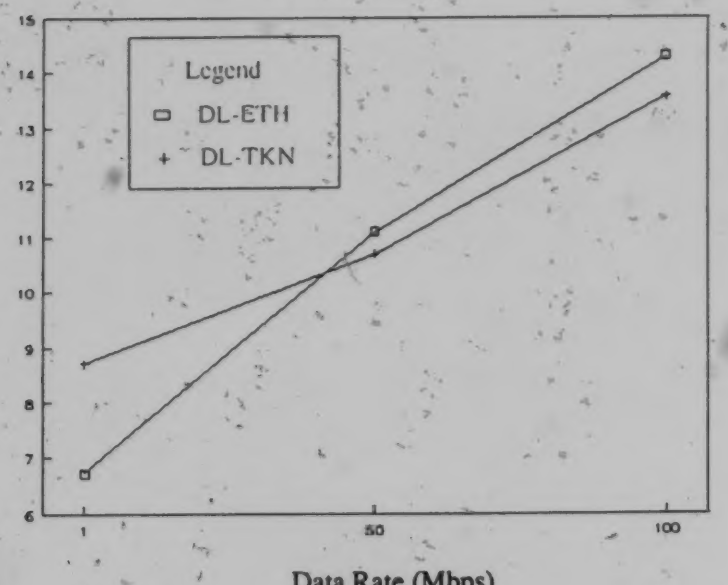


Figure 5.26 Effect of data rate on mean throughput.

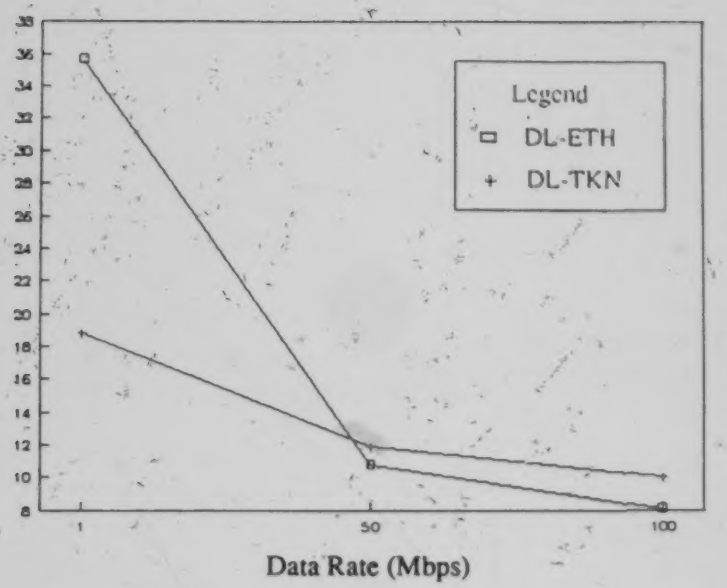


Figure 5.27 Effect of data rate on conflict (%).

5.2.3. Effect of Number Sites

The effect of number of sites on performance of CL over Ethernet and token ring in terms of mean response time of transaction is shown in Figure 5.28. For both network protocols, increasing the number of sites in the system degrades the performance. As mentioned before, as the number of sites increases, the overall rate of arrival of transactions into the system also increases causing network contention. This impact is more critical for DL over Ethernet due to the increasing the probability of collisions between packets. Figures 5.29 and 5.30 show the performance in terms of mean response time of transaction and abortion rate affect by changing of number of sites. The same phenomena as above are observed.

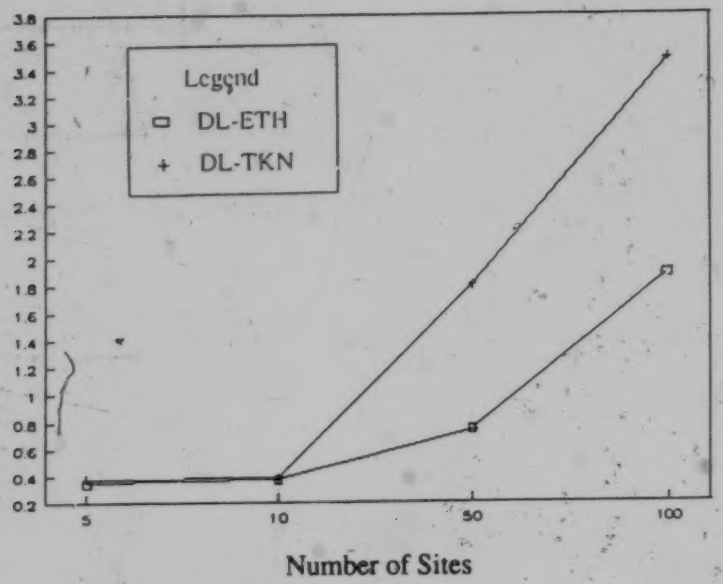


Figure 5.28 Effect of number of sites on response time (sec).

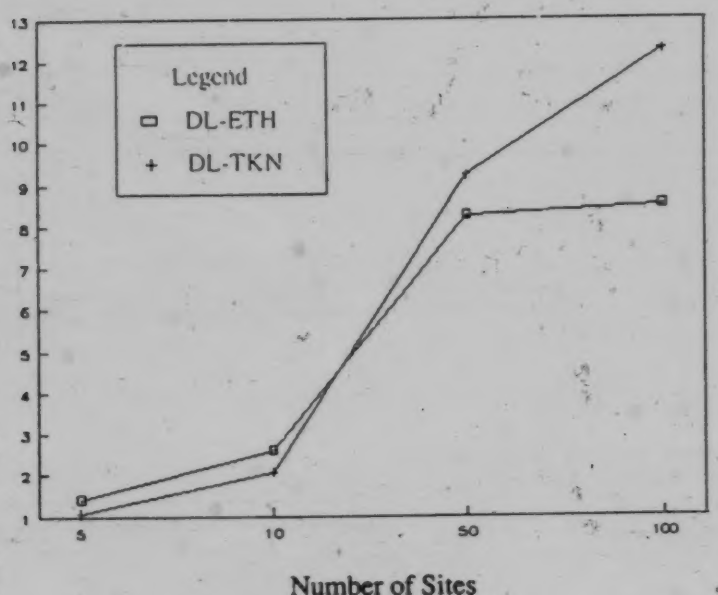


Figure 5.29 Effect of number of sites on mean throughput.

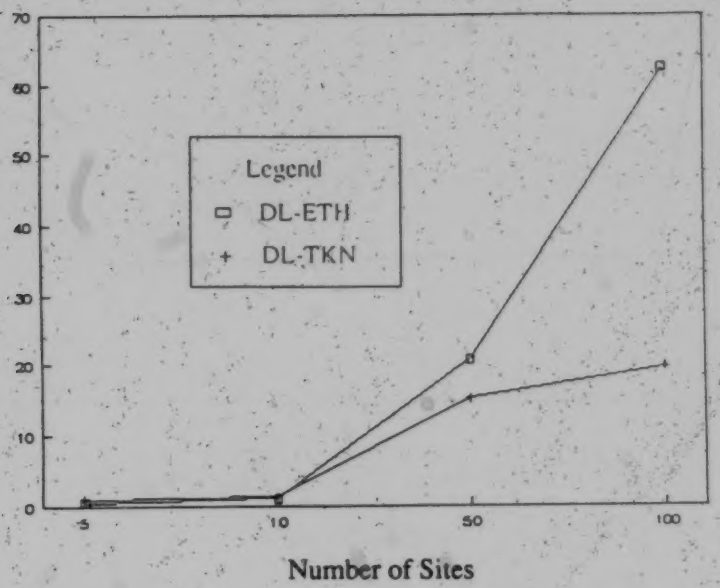


Figure 5.30 Effect of number of sites on conflict (%).

Chapter 6
Implications for DDBS Design

The simulation results in the previous chapter show that token rings have advantages over Ethernet for DL in which the network delay is very critical. Transaction performance of CL over Ethernet is highly variable and will be perceived as much less consistent than one on a ring.

In this chapters, we will discuss how the performance of transaction management in a DDB is affected by the specific features of Ethernet and token ring. Further, we will make an attempt to identify which category of concurrency control and deadlock detection algorithms appear to be more suitable for bus and ring LANs that use CSMA/CD and token access mechanisms, respectively.

6.1. Two-Phase Commit Protocols

A standard implementation guaranteeing atomicity (See Chapter 2) in database has been developed and given the name *two-phase commit*. In two-phase commit, there is one node with the role of taking the final decision of *commit* or *abort* called the *coordinator*, and other nodes that must commit together called the *participants*. The basic idea of the two-phase commit protocol is to determine a unique decision for all participants with respect to committing or aborting a transaction.

As indicated by its name, two-phase commit consists of two phases. During the first phase the coordinator issues *prewrite* commands to all participants, each of which either answers *ready* or *abort* corresponding to "ready to commit" and "not ready", respectively. If a failure occurs during this phase there is no problem since nothing has been stored in the database yet.

During the second phase, the coordinator sends a *write (commit)* or an *abort* command to all participants. All participants will then commit or abort the transaction accordingly. In case of a failure in the second phase, the system must be subjected to a recovery procedure if the transaction is to be committed later.

In these protocols, in the first phase, the coordinator site issues prewrite commands to all participants. In the second phase, depending on the responses received from the participants, the coordinator issues commit or abort commands.

In a WAN, these *prewrite, commit* or *abort* commands have to be sent on each point-to-point link by the coordinator. In a LAN, the broadcast property can be conveniently used to send these to the participating sites. Therefore, with respect to the number of messages that have to be transmitted, 2PC protocols appear to be very suitable for LANs in general. Now we will see how the protocol characteristics of Ethernet and token ring could have an impact on the performance of 2PC protocols.

In response to a *prewrite* command from the coordinator, participants will either respond with *ready* or *about*. If the participants are similar in their processing speeds and the workload they carry, they will attempt to access the network more or less at the same time to transmit their responses. In an Ethernet, this could lead to many collisions. If an Ethernet is heavily loaded, these multiple access attempts could result in long network delays. In such cases, the coordinator could get timed-out if it does not receive all answers within the time-out interval.

A similar scenario could occur when participants attempt to transmit the final *acknowledgement* messages in response to a *global-commit* command from the coordinator in the second phase. Therefore, 2PC protocols implemented in a heavily loaded Ethernet could result in a large number of transaction aborts due to premature time-outs, unless the time-out interval is made very long. In any case, even if most of the transactions are

committed, the overall response time could be unacceptable.

In a token ring environment, even under heavy loads, the delay is bounded and therefore 2PC protocols will perform satisfactorily. A linear type protocol described in [CeP84] appears to be very suitable with appropriate modifications for token ring type networks. This protocol could be modified in the following manner to obtain a much better transaction response time. The modification is necessary to take advantage of the unidirectional nature of message flow in the ring network.

Let there be N sites including the coordinator. For the purpose of explanation, let site 1 be the coordinator.

(a) Coordinator sends a *prewrite* command to its adjacent site (site 2 in this case).

(b) Site 2 decides whether it wants to *commit* or *abort*. If it is ready to commit, the *prewrite* command of coordinator is repeated to the next site (site 3). Otherwise, an *abort* message is sent to the coordinator. In this case, the *prewrite* command of the coordinator need not be sent to remaining sites, as the transaction will be aborted anyway.

All other sites behave in the same manner as described above.

(c) According to the above description, finally the coordinator will either receive an *abort* message (from any one of $N-1$ other sites) or a *commit* message from site N. The *abort* message will indicate that at least one site is not prepared to *commit*. Therefore, the coordinator can decide to *abort* the transaction. On the other hand, the *commit* message received from site N implies that all sites are ready to commit. Now the coordinator can send a global *commit* command for final commitment.

The above modified protocol has the advantage that, with only one propagation round the ring, the coordinator could know whether it could commit or abort the transaction. Oth-

erwise, normally $N-1$ separate messages from $N-1$ sites would have been necessary indicating individually whether each site is prepared to commit or abort. Therefore, while the network is used efficiently, a lower transaction time could be expected. Further, only a minimum amount of processing is required at the coordinator and the participating sites.

6.2. Concurrency Control Algorithms

Two-phase locking (2PL) algorithms used for concurrency control could either be *centralized* or *distributed*. In the case of centralized 2PL algorithms, the broadcast property of a LAN could be used to send messages such as *perform update* for updating replicated copies. Further, the overall response time could be minimized by giving higher priority to central site where *lock tables* are maintained and *lock grant* decisions are made. This is because, the central site needs to access the network more frequently than other sites to send its decisions to the requesting sites.

In Ethernet, priority station operation is not supported. Therefore, other than the broadcast capability and the quicker response at light loads, Ethernet does not offer any other specific advantages for centralized 2PL algorithms. In fact, a heavily loaded Ethernet could make the central site a bottleneck in implementing these algorithms.

In chapter 3, we have mentioned that in Ethernet a packet that has experienced several collisions could suffer a longer delay than a packet that has experienced few or no collisions. Therefore, it is possible for an older transaction to experience longer delays than a younger one, especially if the Ethernet is heavily loaded. This could cause a significant degradation in the performance of timestamp-based concurrency control algorithms. This is because many transactions having smaller timestamp values (i.e. older transactions) may attempt to read (write) data items having larger (i.e. more recent) read

(write) timestamp values. Unless a history is maintained for each of these data items, these transactions have to be restarted. A large number of transaction restarts under heavy load conditions will present an additional load to the system, thus causing more collisions and longer delays. This way, an undesirable cascading effect could build up, resulting in unacceptably high response times and reduced system throughput. This shows that timestamp based algorithms are not suitable for heavily loaded contention type networks.

Token rings can support station priorities. This could be done by allowing preferred nodes to use the token for consecutive packed transmissions. A token timer could be used to prevent a high priority station using the token for an indefinite period of time. Therefore, centralized 2PL algorithms could achieve a better response time even under heavy loads in token ring environments. Token rings are also good candidates for implementing timestamp-based algorithms, because of their bounded delay. Under normal circumstances, even with heavy loads, too many transaction restarts cannot occur in token rings.

Wah and Lien [WaY85] have proposed a concurrency control algorithm that takes advantage of the broadcast property of bus networks. After an initial processing phase, a transaction is distributed to related sites. Each site maintains relevant information in a Broadcast Transaction Table. The broadcasting of information thus allows each site to maintain necessary information for concurrency control. Authors suggest that these broadcasts should be given higher priority to transmit to ensure early distribution of transactions to sites. As we have seen, this is not feasible in Ethernet networks.

6.3. Deadlock Detection/Prevention

Most of the DDB systems use time-out mechanisms as a means of detecting potential deadlocks. This is due to the complexities involved in building and exchanging local and global wait-for graphs [CeP84]. The main problem with time-out schemes is the determination of an optimum time-out interval. If a too short an interval is chosen, many transactions which are not truly in deadlock will be unnecessarily aborted. On the other hand, if the time-out interval is too long, more time will unnecessarily be wasted before deadlocked transactions are aborted.

In Ethernet and other contention schemes where the delay could be unbounded, it will be very difficult to choose an optimum time-out interval. A too short time-out interval could even lead to a cascading effect described in Section 6.2. A long time-out interval would look too short under heavy load conditions. Therefore, time-out schemes for detecting potential deadlocks seem not very appropriate for network schemes where the delay could be unbounded. Deadlock prevention schemes or deadlock detection using local and global wait-for graphs may have to be used under these circumstances.

In token rings or other LAN schemes where the network has a bounded delay, a much more realistic time-out interval can be chosen. Therefore, such networks are more suitable for the implementation of time-out strategies for deadlock detection.

6.4. Network Utilization

In evaluating the overall performance of a DDB system, it is vitally important to consider how effectively the underlying network is being utilized. Therefore, in this section, we will see the utilization aspects of Ethernet and token ring networks.

As already mentioned before (Chapter 3), Ethernet requires a minimum packet length of 64 bytes (64 data bytes) for the operation of the CSMA/CD access protocol.

Therefore, for short messages, it may be necessary to transmit redundant bits. In a DDB system, *abort* or *commit* commands in a 2PC protocol, *locks granted* messages sent by a coordinator in a centralized concurrency control algorithm, etc could typically be short messages. Therefore, in these cases, the effective network utilization (the ratio of useful bits transmitted to the total bits transmitted) will be low. Ethernet also has a maximum size of 1500 data bytes per transmitted frame (or packet). Long messages exceeding this limit has to be broken down in to an integral number of packets. As each packet is associated with a *preamble, source* and *destination addresses* and other fields, more packets imply more overhead in sending some fixed amount of data. Ethernet gives better utilization if majority of messages have an approximate length of 1500 bytes.

Token ring does not have a requirement for a minimum packet length. Therefore, for short messages, redundant information need not be transmitted. The maximum packet length is approximately 4000 bytes in a 4 Mbps token ring. This longer length reduces the overhead associated with transmitting long messages, as now only a fewer number of packets would be necessary. Therefore, in general, token rings appear to give better network utilization than Ethernet. Without knowing the details of an application environment, it would be inaccurate to conclude that token rings give better utilization than Ethernets under all circumstances.

Chapter 7
Conclusions

The design of distributed databases on local computer networks is becoming important with the advent of office information systems and CAD/CAM applications. To improve the performance of a DDB system, the capabilities of the network must also be taken into account in the design. Therefore, in this thesis we studied the effects of the underlying network architecture on the performance of DDB systems. In approaching this problem, we followed two streams: first, we developed and implemented a simulator to study two particular concurrency control algorithms, namely the centralized locking algorithm (CL) and the distributed locking algorithm (DL), with both Ethernet and token ring as the underlying network; secondly, we investigated the effects of LANs on DDB systems in general (Chapter 6).

Our simulation results and investigation show that token rings have advantages over Ethernet type networks in cases where the network delay is very critical such as in DL. The investigation also indicates that timestamp based concurrency control algorithms or algorithms using timeout mechanisms for deadlock detection are more appropriate for a Token ring environment than a contention type network like Ethernet. Simulation results also show token ring to be better in the cases where network loads are heavy and to be more consistent because of the bounded delay property.

Transaction performance of CL over Ethernet is highly variable. High variance will have two impacts on a distributed system. First, it will decrease throughput. That is, of two systems with the same arrival rate, the one with the lower variance will have a lower average response time, all other things being equal. Secondly, it will degrade the

user perception of such a system. Even at low loads, a database running on Ethernet will be perceived as much less consistent than one on a ring. Nonstationary loads will also tend to aggravate problems of CL over Ethernet, by moving the system up its exponential response time curve. Any concurrency control algorithm that uses more than one message per transaction must degrade more rapidly than the network. The amount of communication delay involved in executing a transaction will be directly proportional to the number of messages involved. If it is possible to bundle transactions together in packets, better throughput can be expected, since we can reduce the number of times a given transaction must contend for the network.

We have made certain assumptions about the system which are not totally realistic. These have been mentioned before, but to reiterate, they are the assumptions on no system failure, fully duplicated databases, and no overlapping of CPU and I/O activities. The conclusions reached above are valid only under these assumptions. To reach more realistic conclusions, the experiments need to be repeated in an environment where they are relaxed. One avenue of our current research is along those lines.

There are several other extensions that could be made to the work presented here. In this study, we only studied locking-based concurrency control algorithms. This research could be extended to encompass other types of concurrency control algorithms, such as, timestamp-based and optimistic methods.

On the other hand, it can be extended to encompass other local area network protocols, such as, Expressnet and Fasnet. Finally, it is not difficult to study the performance of different concurrency control algorithms with Ethernet or token ring as underlying network using our simulator.

Bibliography

[AhB88] Mohan L. Ahuja and J.C. Browne, "Performance Evaluation of Two Concurrency Controls Protocols for Distributed Database with Multiversioned Entities," *Proc. 4th Int. Conf. on Data Engineering*, 1988

[AID76] P.A. Alsberg and J.D. Day, " A Principle for Restilient Sharing of Distributed Resources", *Proc. Second Int. Conf. on Software Engineering*, 1976.

[Ans85] ANSI/IEEE Std. 802.3 ISO/DIS 8802/3, *Local Area Networks-CSMA/CD*,1985.

[BaP78] D.Z. Bernstein and N. Goodman, "A Proposal for Distributed Concurrency Control for Partially Redundant Distributed Database System," *Proc. Third Berkeley Workshop on Distributed Data Management and Computer Networks*, 1978, pp. 273-288.

[Ber80] P.A. Bernstein and D.W. Shipman and J.B. Rothnie, "Concurrency control in a system for distributed databases (SDD-1)," *ACM Trans. Database Syst.*, 5(1): pp. 18-51, March 1980.

[BeG81] P.A. Bernsein and N. Goodman, "Concurrency control in Distributed Database Systems," *Computing Surveys*, 13(2): pp. 185-221, June 1981.

[Che81] W.K. Cheng, *Performance Analysis of Update Synchronization Algorithms for Distributed Database*, Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1981.

[CeP84] S.Ceri and G.Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, 1984.

[ELL77] C.A. Ellis, "A Robust Algorithm for Updating Duplicated Database", *Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks*, 1977, pp. 146-158.

[GaC80] G. Gardarin and W.W. Chu, "A Distributed control algorithm for reliably and consistently updating replicated databases," *IEEE Trans. Comput*, C-29, pp. 1060-1068, Dec. 1980.

[Gar79] H. Garcia-Molina, "Centralized control update algorithms for fully redundant distributed databases," *Proc. 1st. Int. Conf. Distrib. Computing Syst.*, 1979, pp. 699-705.

[Gbu90] P. Gburzynski and P. Rudnicki, *The LANSF Protocol Modeling Environment*, Technical Report TR 90-4, Dept. of Computing Sc., Univ. of Alberta, Alberta, Canada, Jan. 1990.

[Gra78] J.N. Gray, "Notes on database operating systems," In *Operating Systems: An Advanced Course, Lecture Notes in Computer Science, Vol. 60*, Springer-Verlag , New York, 1978, 393-481.

[HeY87] Alan R. Hevner and S. Bing Yao, "Querying Distributed Databases on Local Area Networks," *Proc. IEEE.*, 75(5): pp. 563-572, May 1987.

[Lam 78] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System," *Comm. ACM*, 21(7): pp. 558-565, July, 1978.

[Lan78] G. Le Lann, "Algorithms for Distributed Data Sharing Systems Which use Tickets," *Proc. Third Berkeley Workshop on Distributed Data Management and Computer Networks*, 1978, pp. 259-272.

[LiN83] W.K. Lin and J. Nolte, "Basic Timestamp, Multiple Version Timestamp, and Two-Phase Locking", *Proc. 9th International Conf. on Very Large Data Bases*,

1983, pp. 109-119.

[Mal80] T.A. Marsland and S.S. Isloor, "Detection of Deadlocks in Distributed Database Systems", *INFOR* 13(1): pp. 1-20, Feb. 1980.

[MeB76] R.M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM*, 19(7): pp. 395-404, July 1976.

[Obe82] R. Obermack, "Distributed Deadlock Detection Algorithm," *ACM Trans. Database Systems*, 7(2): pp. 187-208, June 1982.

[Ozs85a] M.T. Ozsu, "Performance Comparison of Distributed vs Centralized Locking Algorithms in Distributed Database Systems", *Proc. 5th International Conference on Distributed Computing Systems*, May 1985.

[Ozs85b] M.T. Ozsu, "Modeling and Analysis of Distributed Database Concurrency Control Algorithm Using an Extended Petri Net Formalism, *IEEE Trans. on Software Eng*, SE-11(10): pp. 1225-1240, 1985.

[Ozs91] M.T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, 1991.

[Pap79] C.H. Papadimitriou, "The Serializability of Database Update," *J. ACM*, 26(4): pp. 631-653, Oct. 1979.

[Pap86] C.H. Papadimitriou, *The Theory of Database Comcurrency Control*, Computer Science Press, 1986.

[Rie79] D.R. Ries, *The Effects of Concurrency Control on Database Management System Performance*, Ph.D. Thesis, Department of EECS, University of California at Berkeley, Berkeley, California, 1979.

[Ros86] Floyd E. Ross, "FDDI-a Tutorial," *IEEE Comm. Mag.*, 24(5): pp. 10-17, May 1986.

[Tho79] R.H. Thomas, "A Majority Consensus Approach to Concurrency Control", *ACM Trans. Database Systems*, 6(2): pp. 180-209, 1979.

[WaY85] Benjamin W. Wah and Yao-Nan Lien, "Design of Distributed Database on Local Computer Systems with a Multiaccess Network," *IEEE Trans. on Software Eng.*, SE-11(7): pp. 606-619, July 1985.

Appendix A1
Performance Results

The performance results obtained for the CL and DL algorithms over both Ethernet and Token Ring protocols are listed in this appendix. The results were obtained from simulation runs of about 1000-2000 transactions each after the system has reached its steady state. To give an accuracy of the results, 95% confidence intervals are computed for the mean response times R as:

$$\left[\bar{R} - 1.96\sqrt{\frac{s^2}{n}}, \bar{R} + 1.96\sqrt{\frac{s^2}{n}}\right]$$

where s^2 is the sample variance obtained from the simulation and n is the total number of samples. The experiment was repeated 3 times with independent random number series. Let R_{ij} be the ith observation in the jth run, and let the sample mean and variance for the jth run be denoted by $\overline{R_j}$ and s_j^2, respectively. For that jth run, the estimates are:

$$\overline{R_j} = \frac{1}{n}\sum_{i=1}^{n} R_{ij}$$

$$s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n} [R_{ij} - \overline{R_j}]^2$$

Combining the results of 3 independent measurements gives the following estimates for the mean response time $\bar{R}$, and the variance, s^2, of the population:

$$\bar{R} = \frac{1}{3}\sum_{j=1}^{n} \overline{R_j}$$

$$s^2 = \frac{1}{3}\sum_{j=1}^{n} s_j^2$$

Ethernet				
	CL		DL	
Interarrival Time	Response Time	Confidence Interval	Response Time	Confidence Interval
0.5	19.2	±0.618	2.2	±0.0688
1	13.9	±0.509	1.12	±0.066
5	8.31	±0.401	0.735	±0.0311
10	0.588	±0.0959	0.603	±0.0221
15	0.567	±0.0942	0.47	±0.0125

Table A1.1 Effect of interarrival time on mean response time.

Ethernet			
	CL		DL
Interarrival Time	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
0.5	0.89	0.807	0.0604
1	0.91	0.925	0.0706
5	0.96	0.934	0.0419
10	0.639	0.603	0.0315
15	0.52	0.521	0.0168

Table A1.2 Effect of interarrival time on I/O utilization.

Ethernet			
	CL		DL
Interarrival Time	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
0.5	0.0628	0.0527	0.0163
1	0.053	0.0408	0.015
5	0.0365	0.0221	0.00591
10	0.0305	0.0161	0.00308
15	0.0138	0.0131	0.00225

Table A1.3 Effect of interarrival time on CPU utilization.

Ethernet				
	CL		DL	
Interarrival Time	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
0.5	12.9	0.258	15.1	0.302
1	10.5	0.21	13.9	0.278
5	8.59	0.172	8.26	0.165
10	5.31	0.106	5.01	0.102
15	3.5	0.07	3.34	0.0669

Table A1.4 Effect of interarrival time on mean throughput.

Ethernet		
	CL	DL
Interarrival Time	Mean Conflict	Mean Conflict
0.5	0.969	0.52
1	0.934	0.45
5	0.881	0.208
10	0.0565	0.0733
15	0.0494	0.0593

Table A1.5 Effect of interarrival time on mean conflict.

Ethernet		
	CL	DL
Interarrival Time	Network Utiliz	Network Utiliz
0.5	0.21	0.47
1	0.178	0.42
5	0.126	0.31
10	0.0374	0.226
15	0.0109	0.0921

Table A1.6 Effect of interarrival time on network utilization.

Ethernet				
	CL		DL	
Base set Size	Response Time	Confidence Interval	Response Time	Confidence Interval
2	1.285	±0.0315	0.33	±0.015
5	8.31	±0.401	0.735	±0.0311
10	13.6	±1.03	1.14	±0.0599

Table A1.7 Effect of mean Base set on mean response time.

Ethernet			
	CL		DL
Base set Size	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
2	0.601	0.613	0.0234
5	0.96	0.934	0.0419
10	0.97	0.982	0.0426

Table A1.8 Effect of mean Base set on I/O utilization.

Ethernet			
	CL		DL
Base set Size	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
2	0.0297	0.0241	0.00551
5	0.0395	0.0221	0.00602
10	0.0446	0.0264	0.00502

Table A1.9 Effect of mean Base set on CPU utilization.

Ethernet				
	CL		DL	
Base set Size	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
2	10.4	0.208	10	0.2
5	8.59	0.172	8.26	0.165
10	4.27	0.085	5.45	0.109

Table A1.10 Effect of mean Base set on mean throughput.

Ethernet		
	CL	DL
Base set Size	Mean Conflict	Mean Conflict
2	0.0796	0.0518
5	0.881	0.208
10	1.27	0.448

Table A1.11 Effect of mean Base set on mean conflict.

Ethernet				
	CL		DL	
Database Size	Response Time	Confidence Interval	Response Time	Confidence Interval
10	38.7	±5.48	0.394	±0.205
100	20.8	±3.48	0.629	±0.0314
500	10.51	±1.07	0.716	±0.0373
1000	8.31	±0.401	0.735	±0.0311
1200	7.34	±0.337	0.785	±0.0373
1500	6.6	±0.314	0.868	±0.0451

Table A1.12 Effect of database size on response time.

Ethernet			
	CL		DL
Database Size	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
10	0.0856	0.0845	0.0012
100	0.191	0.219	0.0198
500	0.965	0.937	0.0309
1000	0.96	0.934	0.0419
1200	0.962	0.971	0.0415
1500	0.965	0.92	0.0378

Table A1.13 Effect of database size on I/O utilization.

Ethernet			
	CL		DL
Database Size	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
10	0.00577	0.00307	0.000691
100	0.00785	0.00522	0.00361
500	0.034	0.0212	0.00507
1000	0.0395	0.0221	0.00602
1200	0.0394	0.0229	0.00613
1500	0.034	0.0189	0.00547

Table A1.14 Effect of database size on CPU utilization.

Ethernet				
	CL		DL	
Database Size	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
10	4.26	0.0852	0.982	0.0196
100	5.27	0.165	7.04	0.141
500	7.85	0.177	8.14	0.163
1000	8.59	0.172	8.26	0.165
1200	9	0.18	8.32	0.186
1500	9.87	0.197	8.85	0.197

Table A1.15 Effect of database size on mean throughput.

Ethernet		
	CL	DL
Database Size	Mean Conflict	Mean Conflict
10	1.2	0.847
100	1	0.396
500	0.917	0.237
1000	0.881	0.208
1200	0.737	0.171
1500	0.706	0.134

Table A1.16 Effect of database size on mean conflict.

Ethernet				
	CL		DL	
Number of Sites	Response Time	Confidence Interval	Response Time	Confidence Interval
5	0.233	±0.0336	0.345	±0.033
10	0.502	±0.0461	0.369	±0.0342
50	8.31	±0.401	0.735	±0.0311
100	35.11	±2.53	1.799	±0.0324

Table A1.17 Effect of number of sites on mean response time.

Ethernet			
	CL		DL
Number of Sites	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
5	0.161	0.153	0.0549
10	0.309	0.306	0.0515
50	0.96	0.934	0.0419
100	0.989	0.963	0.0102

Table A1.18 Effect of number of sites on I/O utilization.

Ethernet			
	CL		DL
Number of Sites	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
5	0.0076	0.00336	0.00185
10	0.0154	0.00595	0.00233
50	0.0395	0.0221	0.00602
100	0.0617	0.0523	0.00268

Table A1.19 Effect of number of sites on CPU utilization.

Ethernet				
	CL		DL	
Number of Sites	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
5	1.15	0.23	1.67	0.354
10	2.13	0.213	2.97	0.297
50	8.59	0.172	8.26	0.165
100	9.96	0.096	8.5	0.085

Table A1.20 Effect of number of sites on mean throughput.

Ethernet		
	CL	DL
Number of Sites	Mean Conflict	Mean Conflict
5	0.00399	0.00498
10	0.0276	0.0105
50	0.881	0.208
100	0.914	0.624

Table A1.21 Effect of number of sites on mean conflict.

Ethernet				
	CL		DL	
Date Rate (Mbps)	Response Time	Confidence Interval	Response Time	Confidence Interval
1	22.6	±4.03	1.694	±0.0454
10	8.31	±0.401	0.735	±0.0311
50	3.34	±0.386	0.703	±0.0351
100	1.73	±0.109	0.395	±0.0207

Table A1.22 Effect of date rate on mean response time.

Ethernet			
	CL		DL
Date Rate (Mbps)	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
1	0.901	0.852	0.0383
10	0.96	0.934	0.0419
50	0.977	0.934	0.0416
100	0.985	0.956	0.0538

Table A1.23 Effect of date rate on I/O utilization.

Ethernet			
	CL		DL
Date Rate (Mbps)	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
1	0.0214	0.0372	0.00567
10	0.0395	0.0221	0.00602
50	0.0617	0.0523	0.00968
100	0.0801	0.0675	0.0176

Table A1.24 Effect of date rate on CPU utilization.

Ethernet				
	CL		DL	
Date Rate (Mbps)	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
1	3.1	0.062	6.72	0.134
10	8.59	0.172	9.26	0.185
50	12.07	0.241	11.1	0.228
100	15.2	0.304	14.3	0.286

Table A1.25 Effect of date rate on mean throughput.

Ethernet		
	CL	DL
Date Rate (Mbps)	Mean Conflict	Mean Conflict
1	0.883	0.357
10	0.881	0.208
50	0.814	0.108
100	0.787	0.081

Table A1.26 Effect of date rate on mean conflict.

Ethernet		
	CL	DL
Date Rate (Mbps)	Network Utiliz	Network Utiliz
1	0.207	0.431
50	0.039	0.192
100	0.011	0.083

Table A1.27 Effect of date rate on network utilization.

Token Ring				
	CL		DL	
Interarrival Time	Response Time	Confidence Interval	Response Time	Confidence Interval
0.5	13.6	±0.852	7.99	±0.436
1	11.85	±0.774	4.02	±0.314
5	7.4	±0.621	1.799	±0.0391
10	1.07	±0.209	0.7	±0.0232
15	0.617	±0.114	0.554	±0.0194

Table A1.28 Effect of interarrival time on mean response time.

Token Ring			
	CL		DL
Interarrival Time	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
0.5	0.913	0.816	0.0604
1	0.939	0.93	0.0706
5	0.97	0.987	0.0483
10	0.744	0.758	0.0243
15	0.508	0.516	0.0158

Table A1.29 Effect of interarrival time on I/O utilization.

Token Ring			
	CL		DL
Interarrival Time	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
0.5	0.0647	0.0516	0.0251
1	0.056	0.0413	0.0194
5	0.0396	0.0225	0.00602
10	0.0254	0.00766	0.00227
15	0.0103	0.00516	0.00118

Table A1.30 Effect of interarrival time on CPU utilization.

Token Ring				
	CL		DL	
Interarrival Time	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
0.5	16.2	0.323	26	0.52
1	13.3	0.266	24.8	0.495
5	8.95	0.179	9.22	0.184
10	5.22	0.104	4.93	0.0987
15	3.42	0.0684	3.3	0.0661

Table A1.31 Effect of interarrival time on mean throughput.

Token Ring		
	CL	DL
Interarrival Time	Mean Conflict	Mean Conflict
0.5	0.918	0.189
1	0.888	0.162
5	0.867	0.152
10	0.0744	0.126
15	0.0574	0.0691

Table A1.32 Effect of interarrival time on mean conflict.

Token Ring		
	CL	DL
Interarrival Time	Network Utiliz	Network Utiliz
0.5	0.427	0.887
1	0.415	0.839
5	0.274	0.79
10	0.161	0.466
15	0.0765	0.318

Table A1.33 Effect of interarrival time on network utilization.

Token Ring				
	CL		DL	
Base set Size	Response Time	Confidence Interval	Response Time	Confidence Interval
2	1.053	±0.0522	0.566	±0.0224
5	7.4	±0.621	1.799	±0.0391
10	12.3	±1.37	2.03	±0.048

Table A1.34 Effect of mean Base set on mean response time.

Token Ring			
	CL		DL
Base set Size	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
2	0.63	0.632	0.0222
5	0.98	0.987	0.0383
10	0.926	0.933	0.0407

Table A1.35 Effect of mean Base set on I/O utilization.

Token Ring			
	CL		DL
Base set Size	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
2	0.0296	0.0159	0.00532
5	0.0296	0.0125	0.00591
10	0.0198	0.00804	0.00487

Table A1.36 Effect of mean Base set on CPU utilization.

Token Ring				
	CL		DL	
Base set Size	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
2	10.9	0.218	10.3	0.207
5	8.95	0.179	9.22	0.184
10	5.35	0.107	6.33	0.127

Table A1.37 Effect of mean Base set on mean throughput.

Token Ring		
	CL	DL
Base set Size	Mean Conflict	Mean Conflict
2	0.0496	0.0361
5	0.867	0.152
10	1.06	0.202

Table A1.38 Effect of mean Base set on mean conflict.

Token Ring				
	CL		DL	
Database Size	Response Time	Confidence Interval	Response Time	Confidence Interval
10	39.1	±6.29	0.432	±0.0117
100	20.96	±3.41	0.537	±0.0316
500	11.35	±1.59	0.707	±0.0347
1000	7.4	±0.621	1.799	±0.0391
1200	7.01	±0.57	1.837	±0.0438
1500	6.23	±0.341	1.923	±0.0484

Table A1.39 Effect of database size on response time.

Token Ring			
	CL		DL
Database Size	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
10	0.0313	0.0313	0.00142
100	0.0718	0.073	0.0214
500	0.941	0.948	0.033
1000	0.98	0.987	0.0383
1200	0.989	0.984	0.0402
1500	0.971	0.999	0.0418

Table A1.40 Effect of database size on I/O utilization

Token Ring			
	CL		DL
Database Size	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
10	0.0146	0.00582	0.000706
100	0.0251	0.0114	0.0041
500	0.0292	0.0122	0.00511
1000	0.0296	0.0125	0.00591
1200	0.0309	0.0125	0.00599
1500	0.0313	0.0134	0.00594

Table A1.41 Effect of database size on CPU utilization.

Token Ring				
	CL		DL	
Database Size	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
10	3.67	0.0334	0.964	0.0193
100	3.66	0.0732	6.53	0.131
500	7.32	0.146	7.69	0.154
1000	8.95	0.179	9.22	0.184
1200	9.18	0.184	9.35	0.187
1500	9.87	0.197	9.99	0.199

Table A1.42 Effect of database size on mean throughput.

Token Ring		
	CL	DL
Database Size	Mean Conflict	Mean Conflict
10	1.39	0.894
100	1.16	0.417
500	0.925	0.256
1000	0.867	0.152
1200	0.707	0.123
1500	0.691	0.121

Table A1.43 Effect of database size on mean conflict.

Token Ring				
	CL		DL	
Number of Sites	Response Time	Confidence Interval	Response Time	Confidence Interval
5	0.28	±0.0834	0.377	±0.0328
10	0.598	±0.0836	0.382	±0.0369
50	7.4	±0.621	1.799	±0.0391
100	20.8	±1.43	3.47	±0.172

Table A1.44 Effect of number of sites on mean response time.

Token Ring			
	CL		DL
Number of Sites	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
5	0.174	0.172	0.0591
10	0.281	0.269	0.0565
50	0.97	0.987	0.0383
100	0.993	0.987	0.0119

Table A1.45 Effect of number of sites on I/O utilization.

Token Ring			
	CL		DL
Number of Sites	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
5	0.00414	0.00268	0.00194
10	0.00722	0.00376	0.00197
50	0.0296	0.0125	0.00591
100	0.0261	0.0087	0.00474

Table A1.46 Effect of number of sites on CPU utilization.

Token Ring				
	CL		DL	
Number of Sites	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
5	1.06	0.212	1.42	0.284
10	2.08	0.208	2.61	0.261
50	8.95	0.179	9.22	0.184
100	13.2	0.264	12.27	0.1227

Table A1.47 Effect of number of sites on mean throughput.

Token Ring		
	CL	DL
Number of Sites	Mean Conflict	Mean Conflict
5	0.00333	0.0099
10	0.015	0.013
50	0.867	0.152
100	0.909	0.197

Table A1.48 Effect of number of sites on mean conflict.

Token Ring				
	CL		DL	
Date Rate (Mbps)	Response Time	Confidence Interval	Response Time	Confidence Interval
1	17.7	±1.41	2.97	±0.199
4	7.4	±0.621	1.799	±0.0391
16	6.36	±0.856	0.645	±0.0356
50	4.01	±0.532	0.438	±0.0201
100	2.51	±0.421	0.415	±0.0191

Table A1.49 Effect of date rate on mean response time.

Token Ring			
	CL		DL
Date Rate (Mbps)	I/O Utiliz, at Central Site	I/O Utiliz, at Other Sites	I/O Utiliz, at All Sites
1	0.921	0.186	0.0375
4	0.98	0.987	0.0383
16	0.985	0.993	0.0509
50	0.985	0.983	0.0511
100	0.99	0.985	0.0541

Table A1.50 Effect of date rate on I/O utilization.

Token Ring			
	CL		DL
Date Rate (Mbps)	CPU Utiliz, at Central Site	CPU Utiliz, at Other Sites	CPU Utiliz, at All Sites
1	0.0292	0.0101	0.00389
4	0.0296	0.0125	0.00591
16	0.0362	0.0321	0.00593
50	0.0519	0.0473	0.00675
100	0.0702	0.0518	0.0111

Table A1.51 Effect of date rate on CPU utilization.

Token Ring				
	CL		DL	
Date Rate (Mbps)	Throughput in System	Throughput in One Site	Throughput in System	Throughput in One Site
1	4.49	0.089	8.7	0.174
4	8.95	0.179	9.22	0.184
16	9.73	0.195	9.12	0.182
50	11.32	0.226	10.7	0.214
100	13.06	0.261	13.56	0.271

Table A1.52 Effect of date rate on mean throughput.

Token Ring		
	CL	DL
Date Rate (Mbps)	Mean Conflict	Mean Conflict
1	0.869	0.188
4	0.867	0.152
16	0.585	0.122
50	0.832	0.119
100	0.801	0.1

Table A1.53 Effect of date rate on mean conflict.

Token Ring		
	CL	DL
Date Rate (Mbps)	Network Utiliz	Network Utiliz
1	0.32	0.808
50	0.071	0.27
100	0.053	0.202

Table A1.54 Effect of date rate on network utilization.